CSci 5801: Software Engineering I, Fall 2018
Project 2 – Agile Scrum

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 per deadline. 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 except for the Daily Scum Meeting logs that will go on your team forums on Moodle.

The Problem

Now that you have a voting software system developed that allows an election official to run two different types of elections (i.e. IRV and Open Party Listing), the election officials have decided that they would like some changes made to the system and new functionality added. Instead of using the traditional Waterfall methodology to complete the new changes and functionality, management has decided that Agile Scrum will be used to do one 2-week sprint to see if using an iterative approach to software development is reasonable for this type of project.

Jody, the product owner for the system has put together an initial description of what the election officials and others would like to see in the improved software.

- Jody wants the IRV and Open Party Listing algorithms to work fully as described in project #1.
- The users really like that they can use a CSV file to store the candidates and ballots. They realized that having to type in all of the information such as how many ballots there were, the number of candidates, the number of seats, and the election type (i.e. IRV or Open Party Listing), was too tedious and as much information as possible should be able to be obtained from the file itself. Only the file name should be requested from the user. The file name may be different for each election. The users would like two ways to run the election file. One could be as a command line argument and the other, a prompt for the name of the file. This is the only input that they would like to be prompted for and only if they do not run a command line argument. The developers need to figure out a way to ensure they can turn off the request for file name if a command line argument is passed.
- When prompting for a filename, the user would like a GUI instead of a text prompt. A window should appear and the user should be able to type in a file name or look for a file on disk.
- The election officials have been told by their higher ups that when using IRV, the ballots need to have at least half of the candidates ranked for the ballot to be valid. The ballots will not be invalidated at the point of collection but will need to be done when the election is run with the software system. This is very important since invalidated ballots must be removed from the election. A file needs to be created that stores the invalidated ballots for audit purposes. The name of the file should be invalidated_dateofelection.xxx
- The officials would like to see a table at the of the election showing each round of the IRV and the number of votes that the candidate added/subtracted for that round. The table should be displayed to the screen.
  Example from Wikipedia, en.wikipedia.org/wiki/Instant-runoff_voting,
  (note: we do not have write in candidates in our IRV system so disregard that line):

Candidates		1st Round		2nd Round		3rd Round	
Candidate	Party	Votes	±	Votes	±	Votes	±
Bob Kiss	Progressive	2585	+2585	2981	+396	**4313**	+1332
Kurt Wright	Republican	**2951**	+2951	**3294**	+343	4061	+767
Andy Montroll	Democrat	2063	+2063	2554	+491	0	-2554
Dan Smith	Independent	1306	+1306	0	-1306		
James Simpson	Green	35	+35	0	-35		
Write-in		36	+36	0	-36		
EXHAUSTED PILE		4	+4	151	+147	606	+455
TOTALS[28]		8980	+8980				

- The software should be able to handle more ballots and return the election results in a timely manner. It is reasonable to expect upwards of 100,000 ballots.
- The audit file is stored in case of election contesting and was part of project 1. Now the election officials want, in addition to the audit report, a short report that can be printed and given to the election certification officials. The report should have the date, type of election (i.e. IRV or OPL), the candidates, the number of seats, and the winner(s) of the election. Only the pertinent information is on this report.
- The election officials would like to have the ballots input on the screen directly into the program. They would like to be able to do this for one machine originally and eventually have the voting machines send the information directly to the system. They want a graphical user interface that looks nice. This will mean that a file would become another option for controlling the election and this would provide real-time voting updates. The election official would need a way to tell the program what type of election it is, the number of candidates, etc

Asking Questions

If you have questions about this portion of the project, please see a Shana or a TA during office hours. If you are not able to come to office hours, bring your questions to class.

Deliverables

1) GitHub Team Directory Structure:

```
umn-csci-5801-F18/repo-TeamXXX      : XXX is your team number, all teams have a repository set up
        /Project2                   : Create directory in your team repository to store all work
                /src                : Create directory under Project2 to store all your program files
                                    : Be sure to include your makefile if using C++
                                    : If you have a special directory structure to support your coding, you can
                                    : copy it in the the Project2 directory but you must provide clear instructions
                                    : in the Readme.md
                /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 your javadocs or doxygen documentation here. Use your
                                    : documentation from project1 as a starting point. We need every new :
                                    : method to be documented or changed methods to be updated. You may
                                    : also delete methods due to the new requirements.
                /product_backlogs   : Your original backlog created on Wednesday, November 14th and the backlog
                                    : created during the Sprint Review Meeting will be placed in this directory
                                    : along with all files used in their creation. For example, you may want to
                                    : put user stories and their acceptance criteria in this directory so everyone
                                    : has access to all of the original PBIs.
                /sprint_backlogs    : You will use this directory to document your Sprint Backlogs as you
                                    : progress through the project. For example if you complete a task, you will
                                    : want to document what was completed and what you are then doing (e.g.
                                    : taking another task off of the log.)
                Readme.md           : This is stored in the Project2 directory and should provide instructions for
                                    : us if there is any special handling or issues we should know about.
```

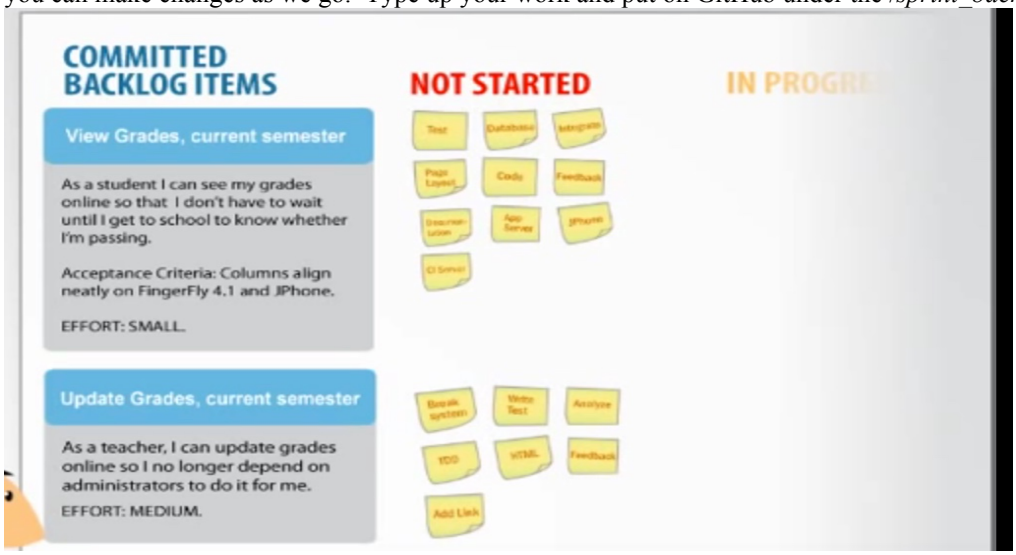
2) Initial Product Backlog: On Tuesday, November 13th during class, we will develop the Product Backlog Items (PBIs) by writing the user stories (with acceptance criteria) and effort estimation. The Product Backlog will be used on

Thursday, November 15th in class to create your Sprint BackLog during the Sprint Planning Meeting. You will put a clean copy of your Product Backlog on GitHub by Friday, November 16th at 11:55 p.m. Name this file: ***InitialProductBackLog.xxx*** where xxx is the file extension of your choice (e.g. .docx, .pdf) You should only have to clean up what we do during class and put your document(s) on GitHub. We will be using sticky notes and paper in class so you can make changes as we go. You will want to type up your work and put on GitHub under the *product_backlogs* directory. Be sure to bring your sticky notes and paper to class on Thursday, November 15th if you have not typed up your results before the Thursday class.

Template for a PBI:

As <user role/persona> I want <what?> So that <why?> Note: We are interested in functionality and not the individual tasks for the PBI. Keep this in mind.
Acceptance Criteria (conditions that have to be fulfilled to ensure the story is complete)
Effort: Small, Medium, Large, Extra Large (estimate of effort and time)

3) Initial Sprint Backlog: On Thursday, November 15th during class, we will create your team's Sprint Backlog using our Product Backlog we developed on Tuesday. We will use the same type of process as was shown in our video, the Sprint Planning Meeting. We will break down the PBIs into tasks and determine how much work can be done in the allotted time for the Sprint. You will put a clean copy of your Initial Sprint Backlog, named ***Initial_Sprint_Backlog.xxx*** on GitHub by Sunday, November 18th at 11:55 p.m. You should only have to clean up what we do during class and put your document(s) on GitHub. We will be using sticky notes and paper in class so you can make changes as we go. Type up your work and put on GitHub under the /*sprint_backlogs* directory.



4) Daily Scrum Meetings: You will need to complete 4 daily standup scrum meetings each week of the sprint. Two (2) of the meetings can be online and **at least two** (2) will be in person. If you meet in person, one of the team members

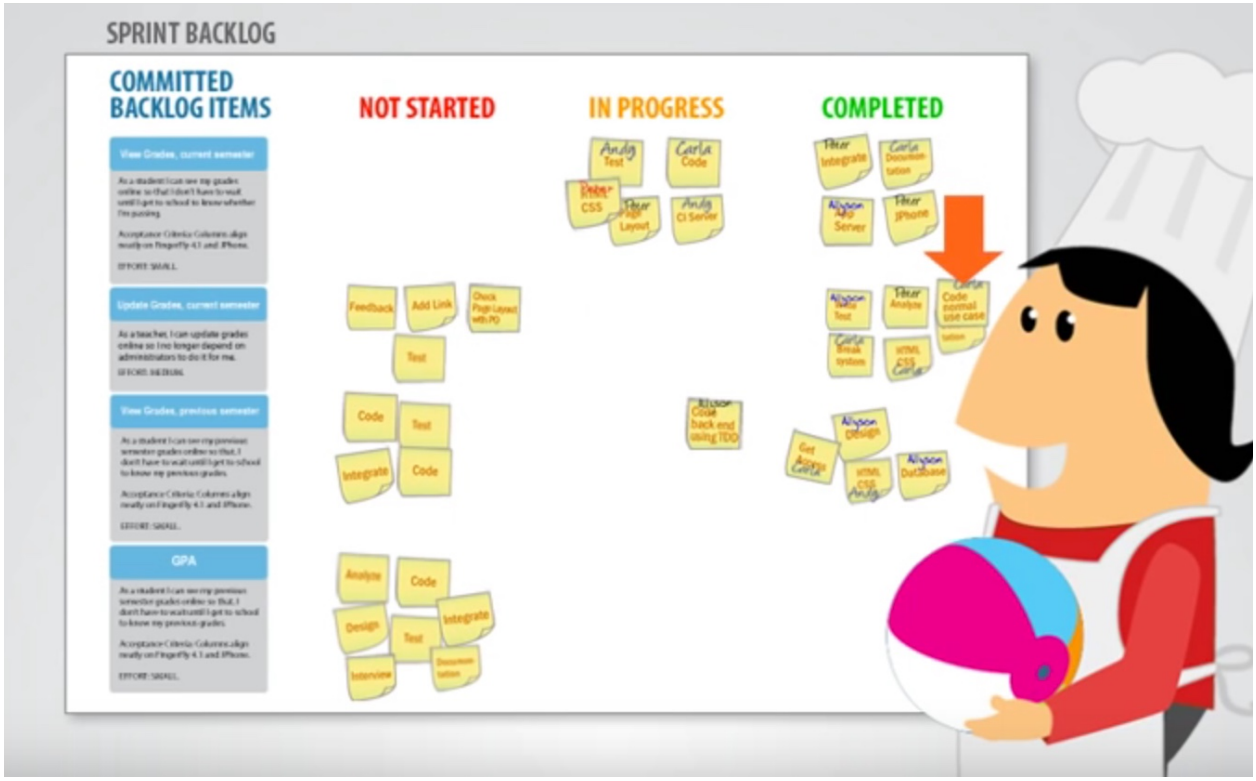
will post the responses to the forum for the group. If online, each person will post their own response on the given day designated as a daily online Scrum meeting day; everyone must post on the same day for it to count as a standup meeting. Take turn posting responses if you meet in person.

- Each team member will answer these questions/prompts at each meeting:

  - Team Member's Name
  - Task that you are working on
  - What did I do yesterday (or since the last Scrum meeting?)
  - What will I do today (before the next Scrum meeting?)
  - What impedes me (blocks my progress, reduces estimates, etc.?) You can state here you need help.
  - Is there a side bar issue to bring up that will be discuss outside of the daily Scrum meeting?
  - Note: Be sure to mention if you updated the Sprint Backlog with new tasks, completed, etc. Are you starting another task or do you need help?

- For each forum posting for a given day, you will state the Type of Meeting (In person/online) and the Scum Meeting # (sequential numbering).

5) Ongoing Sprint BackLog files will need to be kept current on GitHub. We should see updated Sprint Backlogs at **least 2 times a week**. Name the files, *SprintBacklog112618.xxx* where xxx is the extension of the file and the numbers are monthdayyear. If you have not completed anything for that week, make a copy of the older sprint backlog and still add a new file so we can progress through your work.



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

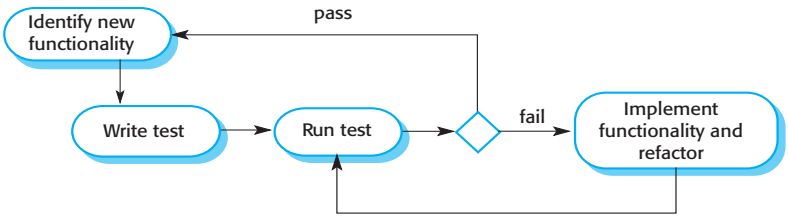
- Correctness and Program Execution: Your code should execute with no syntax or runtime errors and produce correct output. Your program must meet the specifications and function properly for the PBIs that were completed during this Sprint. Remember, if something does not work or is not fully implemented by the end of the Sprint, the item is placed back on the Backlog. The code should only contain the PBIs that were fully coded, tested, and were accepted as done. This encourages using GitHub to maintain versions of the code. Get into the habit of committing work to GitHub throughout the Sprint. The TAs can help you with merging if you find it difficult.
- 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, 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 do 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 you 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 documentation files (i.e. javadocs or doxygen generated files) in their proper locations as defined in the GitHub section of the deliverables.

7) User 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 /Project2/documentation directory on GitHub.

8) Testing: You will be testing your code as you work on your assigned coding tasks. Remember that testing as you go is an integral part of iterative methods. You should identify the new functionality (your task), write your test(s), run the test(s) (and it/they will fail), and then implement the functionality and rerun the tests. The task is only complete when you have passed all tests.

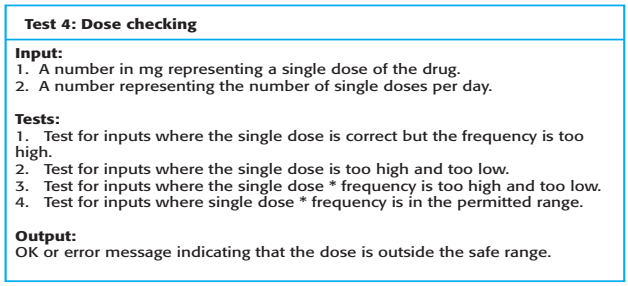


Remember, in Agile Scrum the goal is for the product owner to declare the PBI as finished. If any of your tasks that make up the PBI fail, the entire PBI is moved back to the Product Backlog during the Sprint Review meeting. You will create simple logs as you complete your coding tasks. You can use a testing framework such as JUnit (for Java), Google test framework (for C++), or write your own methods/functions to run your tests. Be sure to give directions in the Readme.md on how to run your tests.

The following is the template to use for your test logs:

The PBI, the Task Description (from Sprint Log) with Unique Testing Number:
Team Member(s) Responsible:
Inputs:
Tests:
Outputs:
Passed or Failed
Date

A partially completed log from our book is found below. Notice that I have asked you to add a few more details.



Test 4: Dose checking

Input:
1. A number in mg representing a single dose of the drug.
2. A number representing the number of single doses per day.

Tests:
1. Test for inputs where the single dose is correct but the frequency is too high.
2. Test for inputs where the single dose is too high and too low.
3. Test for inputs where the single dose * frequency is too high and too low.
4. Test for inputs where single dose * frequency is in the permitted range.

Output:
OK or error message indicating that the dose is outside the safe range.

- Each coding task will have its own testing log. A PBI could have many tasks needed to fully complete it. You should have one file with all testing logs. 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 /Project2/testing directory under your team repository. Your code for the tests will be included in the /Project2/src directory.
- All CSV files used for any testing should be placed in the the /Project2/testing directory. We will move files around as needed when testing your code on a CSE machine.
- Grading: We will be reviewing your logs to determine if your testing was thorough and covered boundary cases and common cases.

Due Dates:

Initial Product Backlog	Uploaded to GitHub under the /product_backlogs directory by Friday, November 16th at 11:55 p.m. – You will clean up the backlog we work on in class
Initial Sprint Backlog	Uploaded to GitHub under the /sprint_backlogs directory by Sunday, November 18th at 11:55 p.m. – You will clean up the backlog we work on in class
Daily Scrum Meetings	At least 2 in person a week and 2 online. Follow instructions online and as written above.
Ongoing Sprint Backlogs	At least 2 files a week should be pushed to GitHub for you ongoing Sprint Backlogs. Follow naming conventions as described above.
Software, Test Logs, Documentation	Everything must be pushed to GitHub by Friday, December 14th at 11:55 p.m.
Final Product Backlog	Uploaded to GitHub under the /product_backlogs directory by Friday, December 14th at 11:55 p.m.

Project Grading Distribution:

Product Backlogs (2 entries)	10%
Sprint Backlogs (Initial and at least 2 per week updates pushed to GitHub)	15%
Daily Scrum Meetings (2 in person per week and 2 online/or in person) – logging Team Forum will be graded	15%
Software (see #6 above)	35%
Testing Logs	15%

Documentation (javadocs or doxygen)	10%