Marissa Alvarez
CMSI 402
2/6/19

Assignment 1

1.1 What are the basic tasks that all software engineering projects must handle?

a. Requirements Gathering
b. High-Level Design
c. Low-Level Design
d. Development
e. Testing
f. Deployment
g. Maintenance
h. Wrap-up

1.2 Give a one sentence description of each of the tasks you listed in Exercise 1.

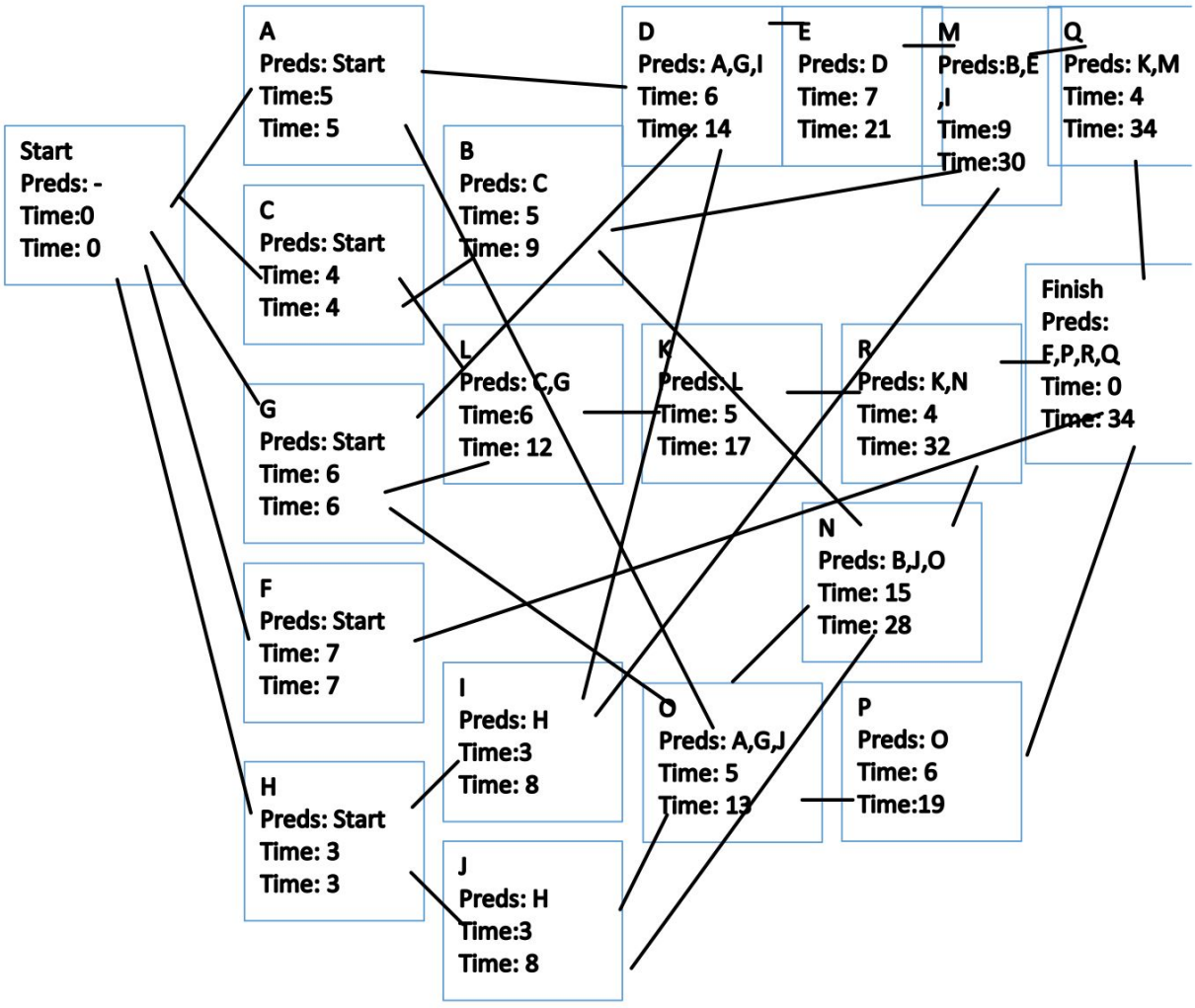
1. Requirements Gathering—Learn the customer's wants and needs.
2. High-Level Design—Describe the major pieces of the application and how they interact.
3. Low-Level Design—Provide more detail about how to build the pieces of the application so that the programmers can actually implement them.
4. Development—Write code to implement the application.
5. Testing—Use the application under different circumstances to try to detect any flaws or bugs.
6. Deployment—Roll out the application to the users.
7. Maintenance—Implement bug fixes, additions, enhancements, and future versions of the program.
8. Wrap-up—Evaluate the project's history to determine what went right and what went wrong so that you can repeat the good things and avoid the bad things in future projects.

2.4 Like Microsoft Word, Google Docs [sic] provides some simple change tracking tools. Go to http://www.google.com/docs/about/ to learn more and sign up [if you do not have an account already]. Then create a document, save it, close it, reopen it, and make changes to it as you did in Exercise 1.

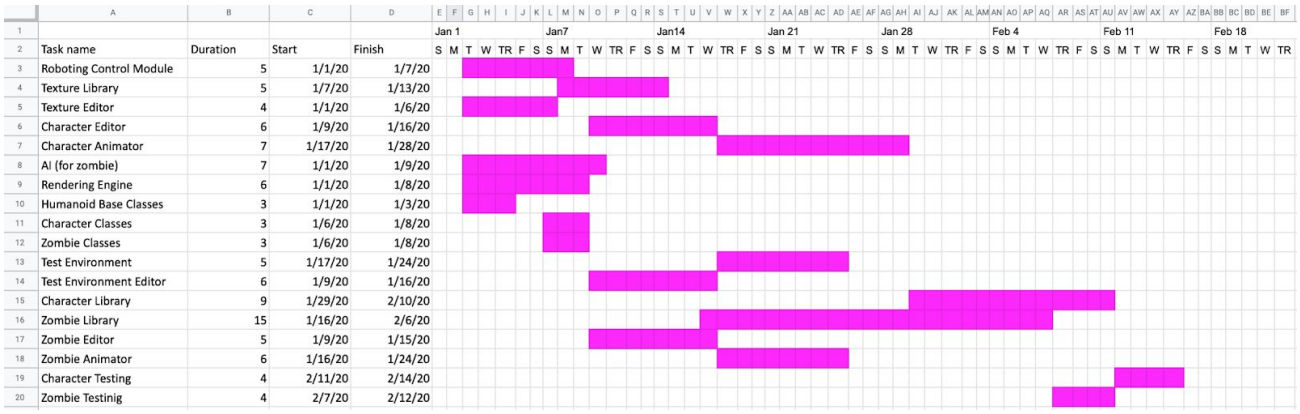
2.5 What does JBGE stand for and what does it mean?

JBGE stands for just barely good enough. It’s the philosophy that you shouldn’t write any more code documentation or comments than absolutely necessary.

3.2 Use critical path methods to find the total expected time from the project's start for each task's completion. Find the critical path. What are the tasks on the critical path? What is the total expected duration of the project in working days?



3.4 Build a Gantt chart for the network you drew in Exercise 3. [Yes, I know, you weren't assigned that one — however, when you do Exercise 2 you should have enough information for this one.] Start on Wednesday, January 1, 2020, and don't work on weekends or the following holidays:



Task name	Duration	Start	Finish
Roboting Control Module	5	1/1/20	1/7/20
Texture Library	5	1/7/20	1/13/20
Texture Editor	4	1/1/20	1/6/20
Character Editor	6	1/9/20	1/16/20
Character Animator	7	1/17/20	1/28/20
AI (for zombie)	7	1/1/20	1/9/20
Rendering Engine	6	1/1/20	1/8/20
Humanoid Base Classes	3	1/1/20	1/3/20
Character Classes	3	1/6/20	1/8/20
Zombie Classes	3	1/6/20	1/8/20
Test Environment	5	1/17/20	1/24/20
Test Environment Editor	6	1/9/20	1/16/20
Character Library	9	1/29/20	2/10/20
Zombie Library	15	1/16/20	2/6/20
Zombie Editor	5	1/9/20	1/15/20
Zombie Animator	6	1/16/20	1/24/20
Character Testing	4	2/11/20	2/14/20
Zombie Testinig	4	2/7/20	2/12/20

3.6 In addition to losing time from vacation and sick leave, projects can suffer from problems that just strike out of nowhere. Sort of a bad version of deus ex machina. For example, senior management could decide to switch your target platform from Windows desktop PSs to the latest smartwatch technology. Or a strike in the Far East could delay the shipment of your new servers. Or one of your developers might move to Iceland. How can you handle these sorts of completely unpredictable problems?

The way one can remedy this is by inserting lost time in the schedule.

- Gaining experience in this type of task
- Breaking large tasks into smaller pieces
- Looking for similarities with tasks you have performed before

3.8 What are the two biggest mistakes you can make while tracking tasks?

1. The biggest mistake you can make while tracking tasks is not taking action when a task slips. At a minimum, you need to pay closer attention to the task so that you can take action if it's in trouble.
2. The second biggest mistake is piling more people on the task and assuming they can cut the total time. Unless the new people have particularly useful expertise, bringing them up to speed may make the task take even longer.

4.1 List five characteristics of good requirements

1. Five characteristics of good requirements are easy to understand, unambiguous, consistent, prioritized, and verifiable.

4.3 Suppose you want to build a program called TimeShifter to upload and download files at scheduled times while you're on vacation. The following list shows some of the applications requirements.

- a. Allow users to monitor uploads/downloads while away from the office. (Business)
- b. Let the user specify website log-in parameters such as an Internet address, a port, a username, and a password. (User, Functional)
- c. Let the user specify upload/download parameters such a number of retries if there's a problem. (User, Functional)
- d. Let the user select an Internet location, a local file, and a time to perform the upload/download. (User, Functional)
- e. Let the user schedule uploads/downloads at any time. (Non Functional)
- f. Allow uploads/downloads to run at any time. (Non Functional)
- g. Make uploads/downloads transfer at least 8 Mbps. (Non Functional)
- h. Run uploads/downloads sequentially. Two cannot run at the same time. (Non Functional)
- i. If an upload/download is scheduled for a time whan another is in progress, it waits until the other one finishes. (Non Functional)
- j. Perform schedule uploads/downloads. (Functional)
- k. Keep a log of all attempted uploads/downloads and whether the succeeded. (Functional)
- l. Let the user empty the log. (User, Functional)
- m. Display reports of upoad/download attempts. (User, Functional)
- n. Let the user view the log reports on a remote device such as a phone. (User, Functioal)
- o. Send an e-mail to an administrator if an upload/download fails more than its maximum retry number of times. (User, Functional)
- p. Send a text message to an administrator if an upload/download fails more than it's maximum retury umber of times. (User, Functional)

All the categories include at least one requirement except for implementation requirements, which is empty. One might need to buy new hardware or network bandwidth to support the application. In that case, there are no implementation requirements. Since one is performing uploads and downloads one may already have the bandwidth therefore there are no implementation requirements.

4.9 Brainstorm this application and see if you can think of ways you might change it. Use the MOSCOW method to prioritize your changes.

A few ways in which someone can change the game is the following:

- Advertising (M)—A phone application typically costs money to install or displays advertising. Currently, the program does neither. It could be modified to display advertising.
- Scoring (S)—Right now you either win or lose. The program could be changed to calculate a score.
- Score keeping (S)—If the program calculates scores, it could keep track of them so that the user can try to beat the previous best score.
- Quick win (C)—The program could allow the user to type a guess for the whole word to get extra points.
- Multiple skill levels (C)—The program could allow users to pick a skill level. An algorithm would use word length and the letters in a word to estimate difficulty.
- Animated win (C)—When the user wins, the program could display an animation.
- Animated loss (C)—When the user loses, the program could animate the finished skeleton (or other picture).
- Report high score (W)—The program could let users report their high scores to a central database so that other users can view them on a web page.
- Animated pictures (W)—The program could display animated pictures that wave, wink, roll their eyes, and so on.
- Time limits (W)—The program could display a countdown. Each correct guess would increase the time available.