CMSI 402 - Assignment 1

- Eileen Choe

Problem 1.1 (pg 12)

> What are the basic tasks that all softare engineering projects must handle?

The basic tasks that all software engineering projects must handle are: requirements gathering, high level design, low level design, development, testing, deployment, maintenance, and wrap up.

Problem 1.2 (pg 12)

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

- Requirements gathering involves determining the customer's wants and needs and transforming them into a requirements document.
- High level design involves determining information about the project architecture at a relatively high level, encompassing tasks such as what platform to use, what data design to use, and how the system will interface with other systems.
- Low level design involves determining how the smaller pieces of the project should work, and interactions/interfaces between them.
- Development is implementing the designs mapped out in the high and low level design in code.
- Testing involves writing functions to ensure that the code is functioning as intended and does not have any bugs.
- Deployment involves "rolling out the software" and includes tasks such as user training and setting up new networks and computers.
- Maintenance involves fixing bugs that are detected by users once the software is deployed.
- Wrap up involves evaluating the project in terms of what went right or wrong, and figuring out how to prevent what went wrong and continue what went right in future projects.

Problem 2.4 (pg 26)

> 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.

Completed exercise.

Problem 2.5 (pg 26)

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

JBGE stands for "just barely good enough" and is a school of thought that you shouldn't provide too much documentation, because you'll waste tie later on updating it when changes to the code are made. The author contends that JGBE is okay as long as the documentation is actually "good enough."

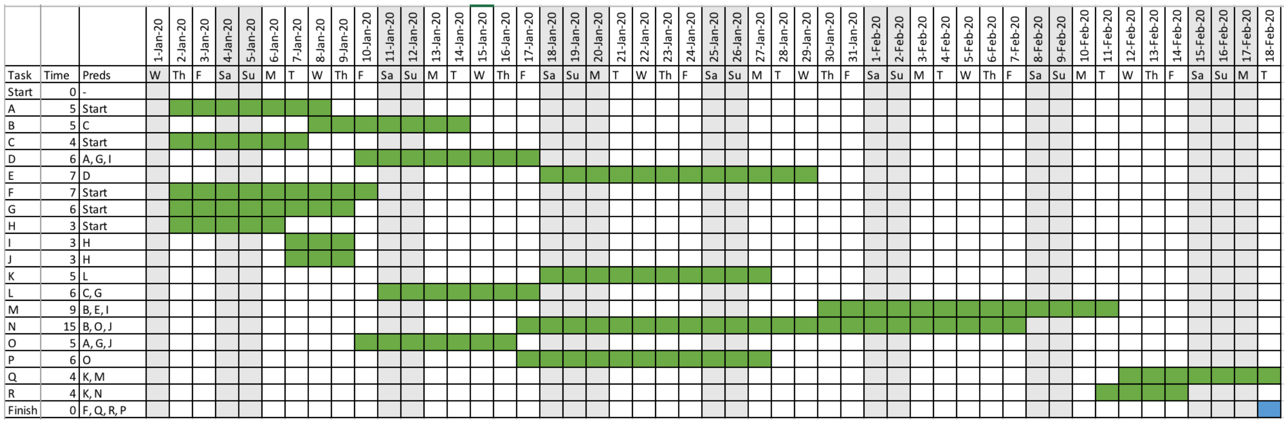
Problem 3.2 (pg 51)

> 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?

The critical path is G => D => E => M => Q. This is 32 working days.

Problem 3.4 (pg 51)

> 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:



			1-Jan-20	2-Jan-20	3-Jan-20	4-Jan-20	5-Jan-20	6-Jan-20	7-Jan-20	8-Jan-20	9-Jan-20	10-Jan-20	11-Jan-20	12-Jan-20	13-Jan-20	14-Jan-20	15-Jan-20	16-Jan-20	17-Jan-20	18-Jan-20	19-Jan-20	20-Jan-20	21-Jan-20	22-Jan-20	23-Jan-20	24-Jan-20	25-Jan-20	26-Jan-20	27-Jan-20	28-Jan-20	29-Jan-20	30-Jan-20	31-Jan-20
Task	Time	Preds	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F
Start	0	-																															
A	5	Start		🟩	🟩	🟩	🟩	🟩	🟩	🟩																							
B	5	C								🟩	🟩	🟩	🟩	🟩	🟩	🟩																	
C	4	Start		🟩	🟩	🟩	🟩	🟩	🟩																								
D	6	A, G, I										🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩														
E	7	D																		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		
F	7	Start		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩																					
G	6	Start		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩																						
H	3	Start		🟩	🟩	🟩	🟩	🟩																									
I	3	H							🟩	🟩	🟩																						
J	3	H							🟩	🟩	🟩																						
K	5	L																		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
L	6	C, G											🟩	🟩	🟩	🟩	🟩	🟩	🟩														
M	9	B, E, I																														🟩	🟩
N	15	B, O, J																	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
O	5	A, G, J										🟩	🟩	🟩	🟩	🟩	🟩	🟩															
P	6	O																	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
Q	4	K, M																															
R	4	K, N																															
Finish	0	F, Q, R, P																															

			1-Feb-20	2-Feb-20	3-Feb-20	4-Feb-20	5-Feb-20	6-Feb-20	7-Feb-20	8-Feb-20	9-Feb-20	10-Feb-20	11-Feb-20	12-Feb-20	13-Feb-20	14-Feb-20	15-Feb-20	16-Feb-20	17-Feb-20	18-Feb-20
Task	Time	Preds	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T
Start	0	-																		
A	5	Start																		
B	5	C																		
C	4	Start																		
D	6	A, G, I																		
E	7	D																		
F	7	Start																		
G	6	Start																		
H	3	Start																		
I	3	H																		
J	3	H																		
K	5	L																		
L	6	C, G																		
M	9	B, E, I	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩							
N	15	B, O, J	🟩	🟩	🟩	🟩	🟩	🟩	🟩											
O	5	A, G, J																		
P	6	O																		
Q	4	K, M												🟩	🟩	🟩	🟩	🟩	🟩	🟩
R	4	K, N											🟩	🟩	🟩	🟩				
Finish	0	F, Q, R, P																		🟦

The project will be finished on February 18th, 2020

Problem 3.6 (pg 51)

> 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?

To handle these unpredictable problems you should expand each task's time estimate by some amount, or add specific tasks to the project to represent lost time or sick time. In addition, a team can carefully plan for approvals, order lead times, and setup.

Problem 3.8 (pg 51)

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

1. The biggest mistake is to ignore problems and hope that you can make up the time later. Instead, you need to assume that you will fall further behind.
2. The second biggest mistake is to pile extra developers on the talk and assume that they can reduce the amount of time needed to finish it. However, adding someone with the appropriate expertise can somtimes help but it takes time for new people to get up to speed on tasks.

Problem 4.1 (pg 82)

> List five characteristics of good requirements.

1. Clear
   - Good requirements are clear, concise, and easy to understand.
2. Unambiguous
   - Good requirements are worded so that you can tell exactly what it requires.
3. Consistent
   - Good requirements are consistent with each other, and don't provide so many constraints that the problem is unsolvable.
4. Prioritized
   - Good requirements are ordered by what's most important.
5. Verifiable
   - Good requirements must be limited in scope and precisely defined so that they can be tested.

Problem 4.3 (pg 82)

> 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 Requirement
>
> - b. Let the user specify website log-in parameters such as an Internet address, a port, a username, and a password.
>
> User & Functional Requirement
>
> - c. Let the user specify upload/download parameters such a number of retries if there's a problem.
>
> User & Functional Requirement

- d. Let the user select an Internet location, a local file, and a time to perform the upload/download.

User & Functional Requirement

- e. Let the user schedule uploads/downloads at any time.

Nonfunctional Requirement

- f. Allow uploads/downloads to run at any time.

Nonfunctional Requirement

- g. Make uploads/downloads transfer at least 8 Mbps.

Nonfunctional Requirement

- h. Run uploads/downloads sequentially. Two cannot run at the same time.

Nonfunctional Requirement

- i. If an upload/download is scheduled for a time whan another is in progress, it waits until the other one finishes.

Nonfunctional Requirement

- j. Perform schedule uploads/downloads.

Functional Requirement

- k. Keep a log of all attempted uploads/downloads and whether the succeeded.

Functional Requirement

- l. Let the user empty the log.

User & Functional Requirement

- m. Display reports of upoad/download attempts.

User & Functional Requirement

- n. Let the user view the log reports on a remote device such as a phone.

User & Functional Requirement

- o. Send an e-mail to an administrator if an upload/download fails more than its maximum retry number of times.

User & Functional Requirement

- p. Send a text message to an administrator if an upload/download fails more than it's maximum retury umber of times.

User & Functional Requirement

> For this exercise, list the audience-oriented categories for each requirement. Are there requirements in each category? [If not, state why not...]

There are no implementation requirements because the end user is not migrating to this system from another system, rather is architecting this system specifically for his or her own use. We assume that the user has everything needed for implementation, since its already being performed manually.

Problem 4.9 (pg 83-84)

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

MOSCOW = Must, Should, Could, Won't

- Must
  - Advertise on the application in order to generate income for the developers.
- Should
  - Implementation of score keeping to encourage players to beat high scores
- Could
  - Include animations to signal transitions to different game states (for example, when a player wins or loses)
  - Improvements to the graphics, such as a different font or color scheme
- Won't
  - Animate graphics