Homework #1

1.1

1. Requirements Gathering
2. High-level Design
3. Low-level Design
4. Development
5. Testing
6. Deployment
7. Maintenance

1.2

1. Requirements gathering is where you figure out what customers want and need and turn that into requirements documents that guide the project.
2. High-level design is where you decide on the platform, architecture, data design,and major interfaces.
3. Low-level design is where you decide on how each piece of the project should work.
4. Development is where you start to refine the low-level designs until they can be implemented in code, then you write the code.
5. Testing is where you test that every aspect of your code is implemented and if there are any errors you send it back for repair and retesiting.
6. Deployment is where you release the software, training users and handling any issues.
7. Maintenance is where you fix the bugs that people find after you release, and also add enhancements and improvements.

2.4

After making both documents and looking at the version history, I noticed it displays a lot of info on the edits made to the document. The right hand panel has a timeline of saved versions with timestamps with a name assigned to each version (the date and time it was saved). It also says who made the changes. WHen you select an old version, edits appear in the document as a different color with added text showing up as colored text while deleted text has a line through it. You can also restore each version easily as there's a button for it.

Github versions are different from this but very similar in a lot of ways. Like docs, it keeps a history of changes over time.lets you restore older versions, shows who made changes, when they made changes, and lets you compare with other versions. The main difference is Github has a much higher barrier of entry. Google docs autosaves, while you must commit each change to github. Github has quite a few other small differences, such as the changes showing line by line, instead of one large block like in google docs, or having different branches while google docs only has one, however the main ideas are similar, Github just offers much more control and technical detail.

2.5

JBGE stands for just barely good enough. It means in terms of documentation that you shouldn't provide too much documentation because you end up wasting a lot of time updating it as you make changes. Many programmers try to do this, but Stephens talks about how most of the people that do, do not understand their work later. He talks about how although documentation is a pain, it is necessary to do and it helps slow you down so you don't "rush off" to a new task.

4.2

a.

Task	**Start**	**Time**	**Finish**
A	0	5	5
B	4	5	9
C	0	4	4
D	6	6	12
E	12	7	19
F	0	7	7
G	0	6	6
H	0	3	3
I	3	3	6

J	3	3	6
K	12	5	17
L	6	6	12
M	19	9	28
N	11	15	26
O	6	5	11
P	11	6	17
Q	28	4	32
R	26	4	30

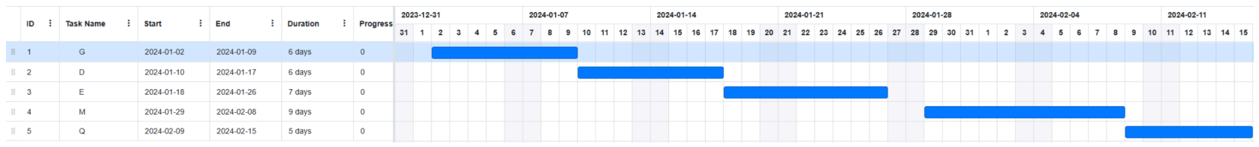
b. There are two critical path methods:

1. G-D-E-M-Q
2. H-I-D-E-M-Q

c. The total expected duration of the project is 32 working days.

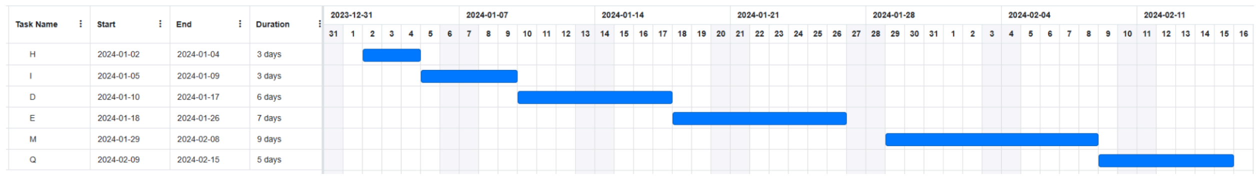
4.4

Gnatt chart 1:



ID	Task Name	Start	End	Duration	Progress
1	G	2024-01-02	2024-01-09	6 days	0
2	D	2024-01-10	2024-01-17	6 days	0
3	E	2024-01-18	2024-01-26	7 days	0
4	M	2024-01-29	2024-02-08	9 days	0
5	Q	2024-02-09	2024-02-15	5 days	0

Gnatt chart 2:



Task Name	Start	End	Duration
H	2024-01-02	2024-01-04	3 days
I	2024-01-05	2024-01-09	3 days
D	2024-01-10	2024-01-17	6 days
E	2024-01-18	2024-01-26	7 days
M	2024-01-29	2024-02-08	9 days
Q	2024-02-09	2024-02-15	5 days

4.6

I can handle these unpredictable problems in a few ways. To start, I can expect these unexpected problems so I am not surprised when they happen. I can also include vacation and sick leave in the schedule towards the end and reassign them as needed. For a larger issue, I can add contingency buffers to the schedule, and especially build flexible project plans without rigid timelines. I can also prepare to reassign work if teams change and allow schedule adjustments when priorities change.

4.8

One of the biggest mistakes you can make while tracking tasks is to ignore the problem if you are behind on schedule and think that you can make up the time later. The second biggest mistake you can make is to pile extra developers on the task and assume they can reduce the amount of time needed to finish.

5.1

1. Good requirements are clear, concise, and easy to understand
2. Good requirements are unambiguous.
3. Good requirements are consistent with each other
4. Good requirements are verifiable
5. Good requirements have priorities

5.3

a. User, Business
b. User, Functional
c. User, Functional
d. User, Functional
e. Nonfunctional
f. Nonfunctional
g. Nonfunctional
h. Nonfunctional
i. Nonfunctional
j. Functional
k. Functional

l. User, Functional
m. User, Functional
n. User, Functional
o. User, Functional
p. User, Functional

There are no requirements in the implementation category. It is only doing tasks like uploading, downloading, logging in, and sending a text/email. implementation requirements are defined as temporary features that are needed to transition to using the new system but will later be discarded. None of these tasks fit in this category.

5.9

Must Have

- Random words so words don't repeat (making sure to leave out offensive words)
- Word hints
- Saving and loading games
- Restarting games at any time
- Easy, medium, and hard modes

Should Have

- Word categories
- Feedback (sounds, vibrations, etc)
- Pausing game
- Screen that shows history

Could Have

- Daily words that are difficult
- Timed mode
- Unlockable characters for the skeleton
- Different colors for the keyboard/background
- 2 player versus mode

Won’t Have (for now)

- Online chat
- Skill tree
- Real purchases

- Story campaign
