Aryaman Ramchandran
Jerahmeel Mendoza
Homework 01

Problem 1.1:

The basic tasks that all software engineering projects must handle typically include:

1. Requirements Analysis and Gathering
2. High-Level Design
3. Low-Level Design
4. Development
5. Testing
6. Deployment
7. Maintenance and Support
8. Postmortem

Problem 1.2:

1. Requirements analysis and gathering involves creating a plan based on customer needs and documenting what the software should do.
2. Developing the high-level design entails creating a blueprint for the software's architecture and components (e.g. choosing which platform to use).
3. Assigning pieces of the high-level design into larger groups to allow them to work in low-level designs. This also means deciding how these pieces will work in combination with each other so it may serve as a guide for implementation.
4. Coding or implementation entails the actual writing of the code. Programmers should rework the low-level design to a point where development is feasible. They should work out bugs in the code as they write it.
5. Testing and quality assurance ensure that the software functions correctly and meets requirements. There should be tests done by people who did not write the code. If the code works, it is implemented into the project and tested again to check if it works in combination with the other code.
6. Deployment is the process of making the software available to users.
7. Maintenance and support involve fixing bugs, making updates, and providing assistance to users.

8. You should evaluate your project afterwards so you can think about what went right and wrong. This make your next project go more smoothly and prevent you from making the same mistakes.

Problem 2.4:

Google Docs tracks changes in text, such as additions, deletions, formatting changes, and comments made within a document. Uses a feature called "Suggesting" mode, similar to "Track Changes" in Microsoft Word. Users can see edits as suggestions, which can be accepted or rejected. Offers version history, showing who made changes and when. Allows real-time collaboration. Multiple users can edit a document simultaneously, and changes are visible instantly. Provides a linear history of changes made over time, allowing users to revert to previous versions of the document.

GitHub tracks changes in source code, including additions, deletions, and modifications to the codebase, across multiple files and directories. Uses commits and pull requests to track changes. Each commit represents a set of changes to the code. Pull requests are used for proposing changes and for peer review before merging into the main branch. Collaboration is centered around pull requests and code reviews. Changes are proposed via branches and merged after review. Offers more complex version control with branching and merging. Supports non-linear development through features like branches, tags, and merges.

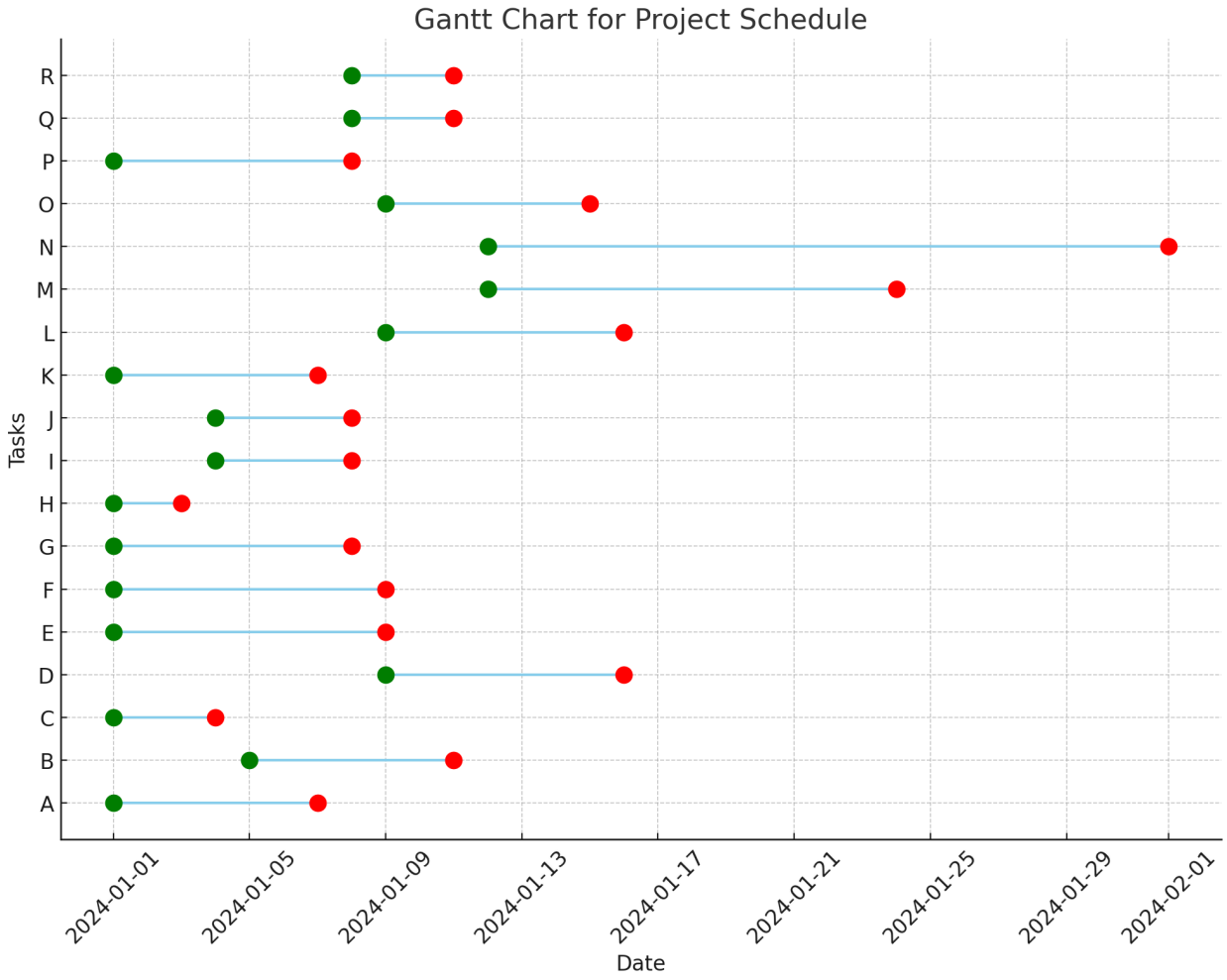
Problem 2.5:

JBGE stands for "Just Barely Good Enough." IT is essentially the concept that documentation for code for be “just barely good enough” or you will waste time documenting as changes to the code are made. It can be effectively implements but can be taken too far and be left with hard to understand code due to lackluster documentation. This rule could be applied to any documentation but in practive mostly occurs in code documentation and comments.

Problem 4.2:

The critical path is the longest path through the network of tasks, determining the shortest time to complete the project. Based on the Critical Path Method (CPM) calculations, the tasks on the critical path are: A, C, E, F, G, H, K, and P. The total expected duration of the project is 24 working days.

Problem 4.4:



Problem 4.6:

Handling completely unpredictable problems in software projects can be challenging. Even if you've tediously planned for possibly lost time and maintained productive task-tracking, there may be times where disaster strikes. In these cases, you want to move towards a more proactive approach like risk management. This involves determining possible risks and their impacts, then planning possible workarounds ahead of time. After analyzing the risks, you can implement it in your project planning.

Problem 4.8:

The two biggest mistakes you can make while tracking tasks are:

1. Ignoring the problem and hoping you can make up for lost time later. You should not assume you can catch up for no reason because this can allow you to fall further and further behind as the weeks go by.
2. Adding more developers to a task in an effort to cut time. While it may be beneficial to add more manpower to a project, it can actually slow down progress if they don't have the right expertise. This is because it takes time for people to get up to speed on a task.

Problem 5.1:

1. Clear: Requirements should be easily understandable and free from confusion or vagueness to all stakeholders.
2. Unambiguous: Every requirement must have only one interpretation, ensuring that all parties have the same understanding.
3. Consistent: Requirements should not conflict with each other and should maintain harmony throughout the document.
4. Prioritized: They should be ranked or ordered to indicate the relative importance or urgency, guiding the development process.
5. Verifiable: Each requirement must be testable, allowing it to be objectively determined whether it has been met.

Problem 5.3:

User Requirements (what the user directly interacts with or experiences):

- a. Allow users to monitor uploads/downloads while away from the office.
- b. Let the user specify website log-in parameters such as an Internet address, a port, a username, and a password.
- d. Let the user select an Internet location, a local file, and a time to perform the upload/download.
- e. Let the user schedule uploads/downloads at any time.
- l. Let the user empty the log.
- n. Let the user view the log reports on a remote device such as a phone.

Business Requirements (needs of the organization using the software):

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

- p. Send a text message to an administrator if an upload/download fails more than its maximum retry number of times.

Functional Requirements (specific behaviors or functions of the system):

- c. Let the user specify upload/download parameters such as the number of retries if there's a problem.
- j. Perform scheduled uploads/downloads.
- k. Keep a log of all attempted uploads/downloads and whether they succeeded.
- m. Display reports of upload/download attempts.

Nonfunctional Requirements (criteria that judge the operation of the system, rather than specific behaviors):

- f. Allow uploads/downloads to run at any time.
- g. Make uploads/downloads transfer at least 8 Mbps.
- h. Run uploads/downloads sequentially. Two cannot run at the same time.
- i. If an upload/download is scheduled for a time when another is in progress, it waits until the other one finishes.

Implementation Requirements (not mentioned in the list):

- The provided list does not explicitly include implementation requirements, which typically relate to how the system should be developed, deployed, or maintained.

Problem 5.9:

Must:

1. Move “New Game” button: too close to buttons and misclicks can cause progress to be lost.
2. Differentiate colors of target word, letters to choose from, and “New Game” button.

Should:

3. Add counter to keep track of how many mistakes player made and how many mistakes they have left.
4. Larger, nicer image of Mr. Bones.

Could:

5. Underline the word.
6. Gray out spaces in the target word that haven’t been filled out yet.

Wont:

7. Change background to more appealing image.
8. Add ads.