Elijah Senior
Assignment 1

1.1
All projects must have a plan, high-level design, low-level design, development, deployment, maintenance, and testing.

1.2

Requirements Gathering: Define what the software should do based on customer needs.

High-level Design: Establish a broad structure and overall architecture.

Low-level Design: Define implementation details for individual components.

Development: Write and implement the actual code for the project.

Testing: Verify that the software meets requirements and is free of defects.

Deployment: Release the software to users in a production environment.

Maintenance: Address bugs, update features, and ensure continued operation.

Wrap-up: Final evaluation of the project and documentation of lessons learned.

2.4

Google Docs provides a simple, nice and easy way to track document edits, it can be used for individuals, small teams, and even large organizations.

Google Docs' version history is linear, showing a chronological list of document edits with timestamps and contributors.

GitHub, in contrast, allows for non-linear versioning, where branches enable parallel development before merging changes.

GitHub tracks changes at the code level.

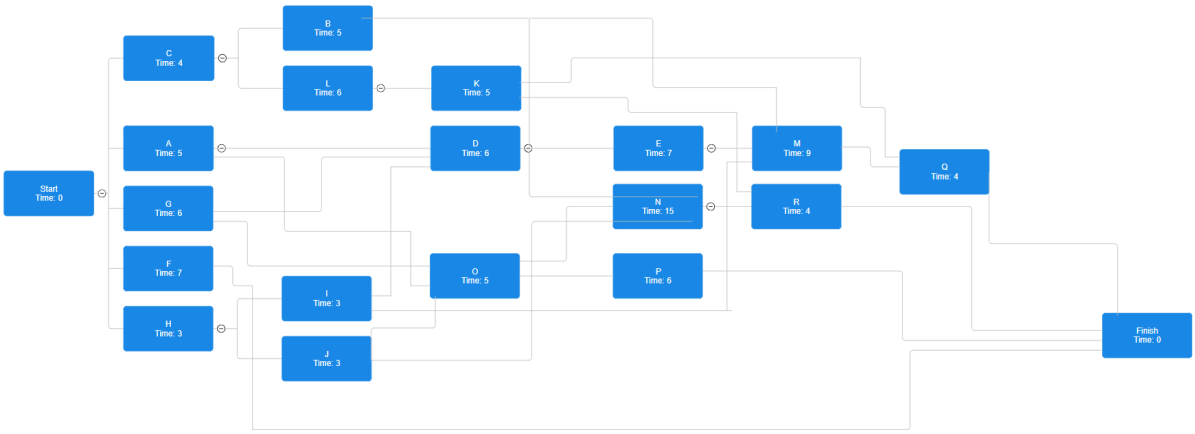
2.5

JBGE means "Just Barely Good Enough."

Documentation and code comments should be minimal yet still easy to read and understand..

4.2

(Eren & I worked on this together)



a. Total expected time is 32 days
b. Critical Path: D, E, M, Q
c. Total expected time in working days: 32 days

To calculate, use a PERT chart and sum the longest sequence of dependent tasks.

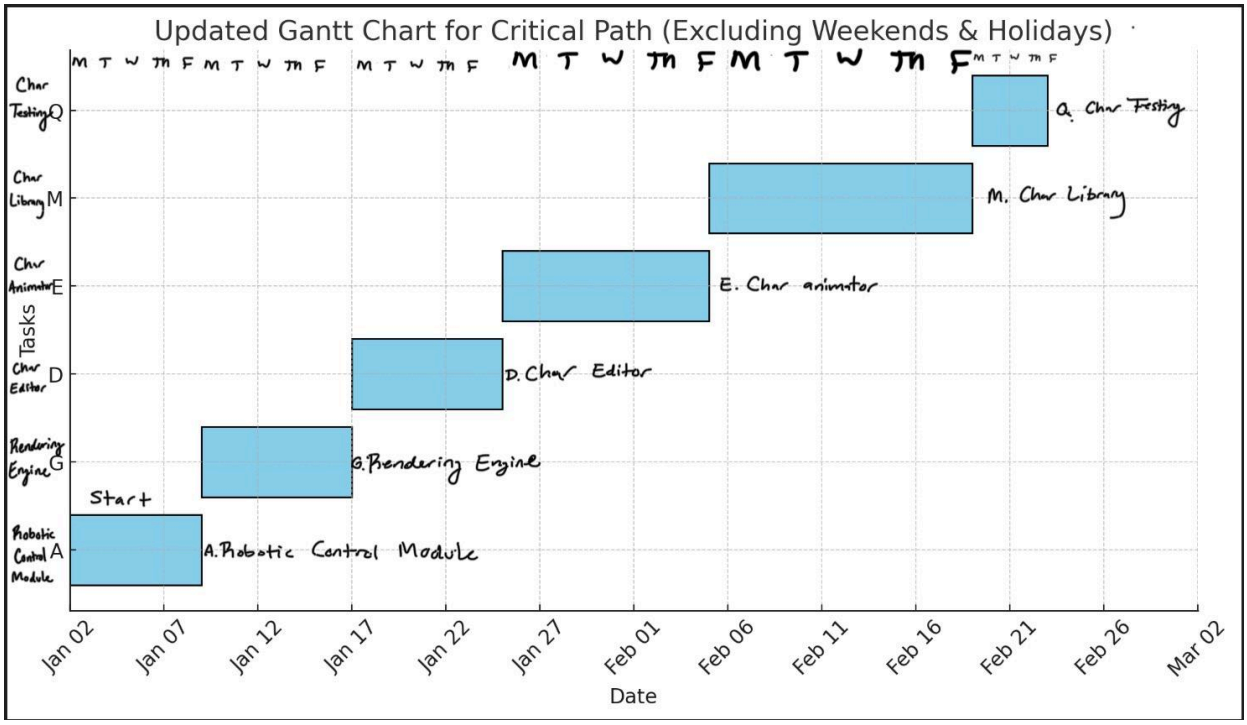
Example from the book: The critical path included tasks Start ➢ A ➢ B ➢ C ➢ D ➢ Finish, totaling 15 days.

4.4

A Gantt chart visually represents the project's timeline.

It should start on Wednesday, January 1, 2024, excluding weekends, and observe specified holidays.

(Eren & I worked on this together)



4.6

Utilize risk management by finding risks, estimating their impact, and planning for things to not go your way..

Always have backup plans, and alternative approaches, and delay non-critical tasks to adapt to sudden issues like changes.

4.8

Failing to adjust for delays: Not updating the schedule when tasks slip. As well as, adding too many developers to a late project.

5.1

Clear and unambiguous: Must be precise and well-defined.

Testable: Should allow verification through testing.

Feasible: Must be achievable given time and resource constraints.

Necessary: This should align with the core project goals.

Prioritized: This should indicate the importance and order of implementation.

5.3

Functional requirements (e.g., file uploads, scheduling).

Security requirements (e.g., authentication parameters).

Performance requirements (e.g., speed constraints).

Usability requirements (e.g., log viewing on remote devices).

Reliability requirements (e.g., retry limits and failure notifications).

Some categories might be missing, like legal compliance or accessibility.

5.9

Should-have: better UI

Could-have: Customization options (themes, sounds).

Won’t-have: Unnecessary features like multiplayer for an initial release.