Daniel Sjarif

B.J. Johnson

CMSI 402

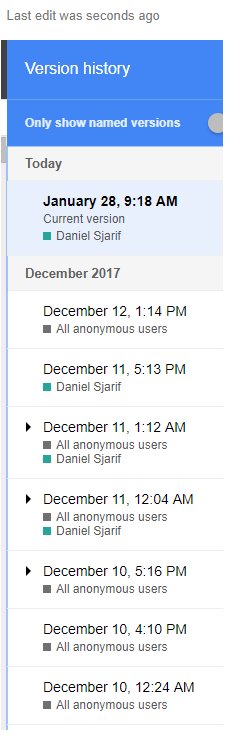
**Problem 1.1**

Requirements Gathering, High-level Design, Low-level Design, Development, Testing, Deployment, Maintenance, Wrap-Up

**Problem 1.2**

* Requirements Gathering: Figure out what the customer wants
* High Level Design: At a high level, describe the major pieces of the application and how they interact
* Low-Level Design: Get into the details about how to specifically build the pieces of the application so the programmers can implement it
* Development: The physical act of writing code to implement the application
* Testing: Stress test the application under different circumstances to try to detect and prevent flaws or bugs before deployment
* Maintenance: Bug fixes, additions, enhancements, adding future versions of the program
* Wrap-up: Look at everything that’s been done in the projects development history to figure out what went right and what went wrong to learn from each project to better development of future projects

**Problem 2.4**



**Problem 2.5**

Just Barely Good Enough means don’t write any more code documentation or comments than necessary

**Problem 3.2**

The critical path passes through the tasks G, D, E, M, and Q. The total expected length of working days is 32. The tasks on the critical path are: Rendering Engine, Character editor, Character animator, Character Library, Character Testing.

**Problem 3.4**

**\***See last page

**Problem 3.6**

Treating these *deus ex machina* problems are not different than handling an unexpected sick leave (which I think can fall under the same category of problems anyways). One can just add tasks at the end of the schedule to account for completely unexpected problems and when one of these problems does occur, insert its lost time into the schedule.

**Problem 3.8**

The biggest mistake one can make while tracking tasks is not taking action when a task slips. It is the bare minimum to pay very close attention to the task so that one can take action as soon as the task is in trouble. The second biggest mistake is piling more people on the task in a bid to save time thinking more people = less time on a certain task. Unless some of those new people have specific abilities, bringing them up to speed may take the task even longer.

**Problem 4.1**

Clear (easy to understand), unambiguous, consistent, prioritized, verifiable

**Problem 4.3**

1. Business
2. User, functional
3. User, functional
4. User, functional
5. Nonfunctional
6. Nonfunctional
7. Nonfunctional
8. Nonfunctional
9. Nonfunction
10. Functional
11. Functional
12. User, functional
13. User, functional
14. User, functional
15. User, functional

All the categories have at least one requirement except for implementation requirements, which is empty. New hardware to support the application may be required, however, if downloads and uploads are already being performed then no further implementations are required.

**Problem 4.9**

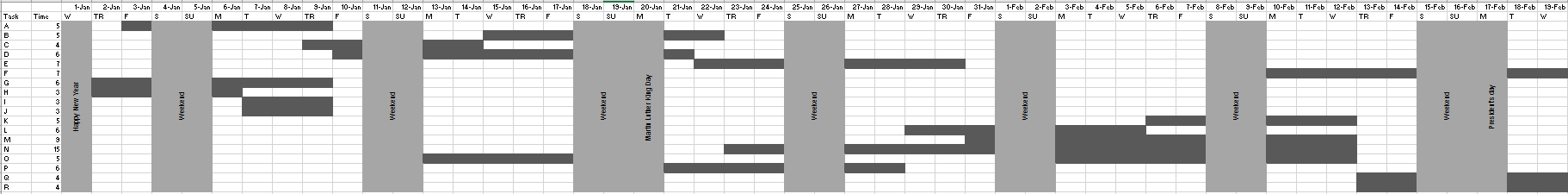
MOSCOW: Must (needed) Should (Important) Could (desirable) Won’t (completely optional).

The musts include: Advertising because for an application to survive they either normally charge a fee to download or they display ads, which this doesn’t do currently. Monetization is important for applications’ survival.

The Shoulds include: Scoring and Score Keeping, high scores; having a simple win and lose game could go stale, having scores and keeping track could incentivize the user to use the application more than they would’ve to try to beat their previous scores.

The Coulds: Different fonts, themes, etc. Would be nice and add some variety to the screen. Skill levels would add variety and challenge players.

The Wonts: Online High scores, to compete with friends. Different languages, for users from different countries. Different ways of interacting rather than just typing (drag and drop etc.).

**3.4**