Ben Davis, Emily Shoji, Nico Pidlaoan
CMSI 402
HW1

Problem 1.1, Stephens page 12

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

- Requirements Gathering
- High-Level Design
- Low-Level Design
- Development
- Testing
- Deployment
- Maintenance
- Wrap-up

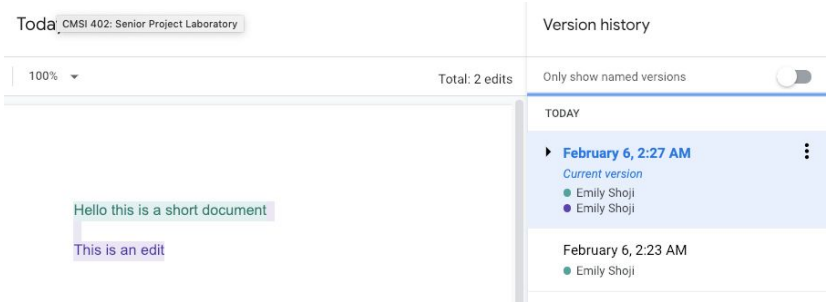
Problem 1.2, Stephens page 12

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

- Requirements Gathering gives the project direction by telling project members what they should be doing how long they should be doing it and most important
- High-Level Design is what major parts the project needs
- Low-Level Design is how the project works and the interactions between the major parts
- Development - programmers get to work define how exactly the low level interactions are defined.
- Testing - Making sure that the code is correct and works as expected for all cases.
- Deployment - How the software will be shared to the public.
- Maintenance is about fixing new bugs as users find them.
- Wrap-up is about evaluating the problem to discuss what went right, what went wrong, and how mistakes can be prevented in the future, while successes are maximised.

Problem 2.4, Stephens page 26

Like Microsoft Word, Google Docs [sic] provides some simple change tracking tools. Then create a document, save it, close it, reopen it, and make changes to it as you did in Exercise 2.



Problem 2.5, Stephens page 26

What does JBGE stand for and what does it mean?

- JBGE stands for “Just barely good enough”. It means that in software documentation, you should provide just enough to be informative. Too much can be a problem if you make changes to the code and have to change the documentation.

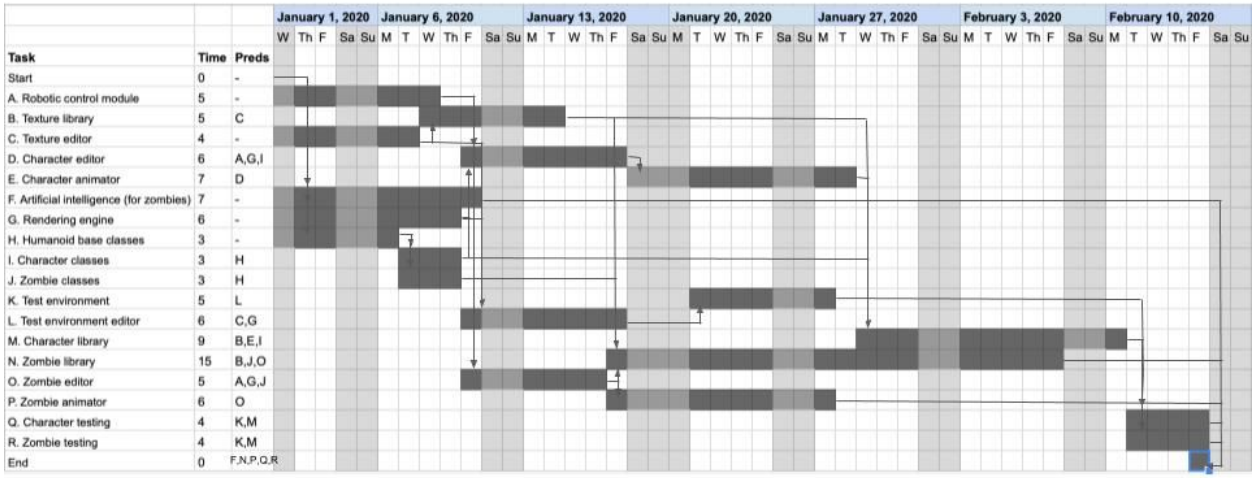
Problem 3.2, Stephens page 51

Use critical path methods to find the total expected time from the project's start for each tasks 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
- Total expected duration is 6+6+7+9+4 = 32 days

Problem 3.4, Stephens page 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:



Task	Time	Preds
Start	0	-
A. Robotic control module	5	-
B. Texture library	5	C
C. Texture editor	4	-
D. Character editor	6	A,G,I
E. Character animator	7	D
F. Artificial intelligence (for zombies)	7	-
G. Rendering engine	6	-
H. Humanoid base classes	3	-
I. Character classes	3	H
J. Zombie classes	3	H
K. Test environment	5	L
L. Test environment editor	6	C,G
M. Character library	9	B,E,I
N. Zombie library	15	B,J,O
O. Zombie editor	5	A,G,J
P. Zombie animator	6	O
Q. Character testing	4	K,M
R. Zombie testing	4	K,M
End	0	F,N,P,Q,R

Target End Date: February 14, 2020

Problem 3.6, Stephens page 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?

- Expand each tasks time estimate by a certain amount.
- Add specific tasks to project to make up for lost time.
- Have placeholder tasks set aside for sick time or extra time needed due to other circumstances

- Plan time for approvals, order leads, and setup.

Problem 3.8, Stephens page 51

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

- Ignore problem and hope you can make it up later.
- Piling extra developers on a task and assume that can reduce time to finish it.

Problem 4.1, Stephens page 82

List five characteristics of good requirements.

1. Clear
2. Unambiguous
3. Consistent
4. Prioritized
5. Verifiable

Problem 4.3, Stephens page 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 — **user requirement**
b. Let the user specify website log-in parameters such as an Internet address, a port, a username, and a password — **functional requirement**
c. Let the user specify upload/download parameters such a number of retries if there's a problem — **functional requirement**
d. Let the user select an Internet location, a local file, and a time to perform the upload/download — **functional requirement**
e. Let the user schedule uploads/downloads at any time — **user requirement**
f. Allow uploads/downloads to run at any time — **user 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 when another is in progress, it waits until the other one finishes — **nonfunctional requirement**
j. Perform schedule uploads/downloads — **user requirement**
k. Keep a log of all attempted uploads/downloads and whether they succeeded — **functional requirement**
l. Let the user empty the log — **functional requirement**
m. Display reports of upload/download attempts — **functional requirement**
n. Let the user view the log reports on a remote device such as a phone — **functional requirement**
o. Send an email to an administrator if an upload/download fails more than its maximum retry number of times — **functional requirement**

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

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

Problem 4.9, Stephens page 83-83

Figure 4-1 shows the design for a simple hangman game that will run on smartphones. When you click the New Game button, the program picks a random mystery word from a large list and starts a new game. Then if you click a letter, either the letter is filled in where it appears in the mystery word, or a new piece of Mr. Bones's skeleton appears. In either case, the letter you clicked is grayed out so that you don't pick it again. If you guess all the letters in the mystery word, the game displays a message that says, "Congratulations, you won!" If you build Mr. Bones's complete skeleton, a message says, "Sorry, you lost." Brainstorm this application and see if you can think of ways you might change it. Use the MOSCOW method to prioritize your changes.

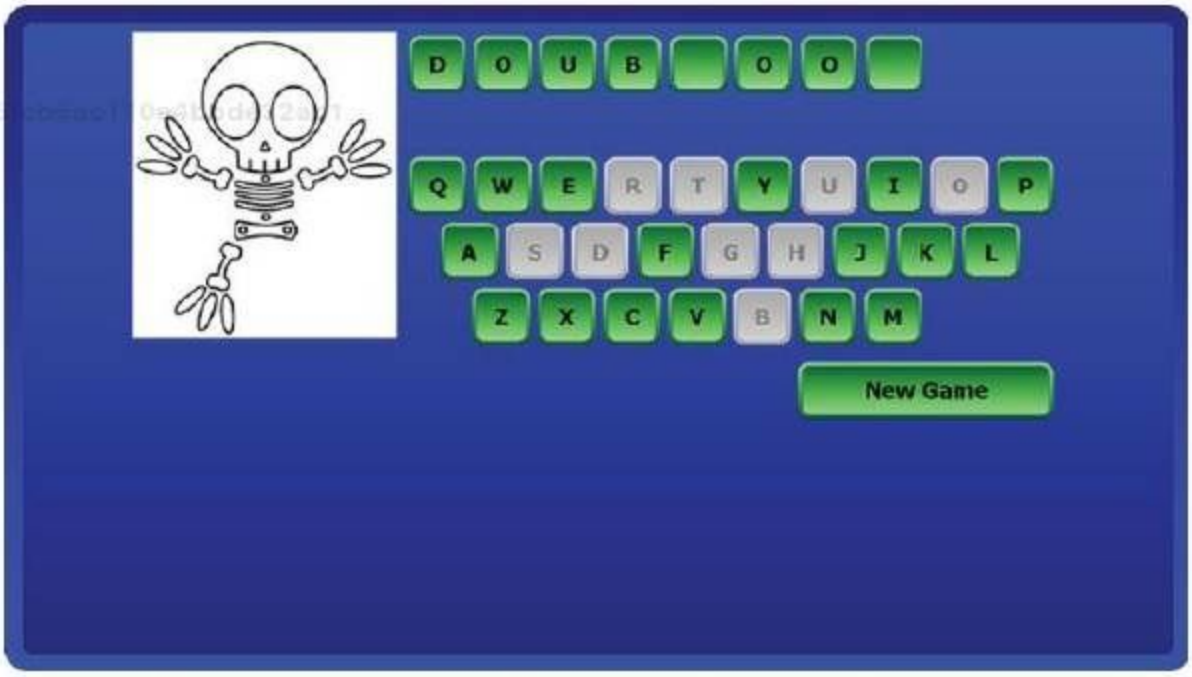


FIGURE 4-1: The Mr. Bones application is a hangman word game for Windows Phone.

Using the MOSCOW method, we can segment the features that could be included in this application into four categories:

- *Must* — This category includes exactly what is shown in the picture, plus the messages that indicate when the player wins or loses. This will suffice for the bare minimum of a hangman game.
- *Should* — Players should definitely have a way of tracking their scores over time, seeing their overall high score, fastest guess, etc.

- *Could* — A multiplayer variation of this game could allow multiple users to play against each other, but this isn't a vital feature from the get-go.
- *Won't* — This is strictly a hangman app for now, so users are not promised any other word games or other anything else unrelated to hangman.