CMSI 4072 Senior Project ll
Homework 01
Christina Choi & Troy Womack-Henderson

1.1 [Stephens page 13]
What are the basic tasks that all software engineering projects must handle?

1). Figuring out the requirements
2). High Level Design
3). Low-Level Design
4). Development
5). Testing
6). Deployment
7). Maintenance
8). Wrap-Up

1.2 [Stephens page 13]
Give a one sentence description of each of the tasks you listed in Exercise 1.1

1). Determine the consumers' wants and needs and turn it into a requirement document - informing customer what they're getting and project members what they're building.
2). Decide what platform you're going to use, what data design to use, and interfaces with other systems
3). Write out how each piece of the project should work.
4). Start writing code, refining the low-level designs as you go and removing bugs as you go.
5). Developers test their own code and then people who didn't write the code test it. Then add the code to the rest of the project and test the whole project.
6). Roll out the software, which may include adding new computers for the back-end database, a new network, etc.
7). Fix bugs if users find any bugs.
8). Perform a post-mortem: evaluate what went right vs. wrong and how to ensure things that went right occur more often in the future as well as prevent what went wrong.

2.4 [Stephens page 27]

Like Microsoft Word, Google Docs [sic] provides some simple change tracking tools. Go to http://www.google.com/docs/about/ to learn more and sign up [if you do not have an account already]. Then create a document, open the File menu's Version History submenu, select Name Current Version, and name the file 'Version 1'. Make some changes and repeat the preceding steps to name the revised document 'Version 2'. Now open the File menu's Version History submenu again but this time select See Version History. Click the versions listed on the right to see what changed between versions. Document what you've noticed about the information you see, and how the differences between versions are displayed.

In each version, it highlights the changes you have made aka, the differences between versions. It also displays the date for each version.

2.5 [Stephens page 27]

What does JBGE stand for and what does it mean?

It stands for "Just Barely Good Enough". The idea is that for code documentation and comments, if you provide too much, you end up wasting a lot of time.

Use the following table of data for Exercises 3.2 and 3.4.

Task	Time (Days)	Predecessors
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, N

4.2 [Stephens page 78]
Use critical path methods to find the total expected time from the project's start for each task's 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?

Critical Path: G -> O -> N -> P = 33

Tasks: Rendering engine, Zombie editor, Zombie library, Zombie animator

Time = 33 days

4.4 [Stephens page 78]
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, 2024, and don't work on weekends or the following holidays:

Holiday	Date
New Year's Day	January 1
Martin Luther King Day	January 20
President's Day	February 17

https://docs.google.com/document/d/13ysjTGYne9ssmekYOcN5x1AGwj1wYLIKJUIorhMN988/edit?usp=sharing

(link to Gantt Chart ^^)

4.6 [Stephens page 79]
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 PCs to the latest smartwatch technology. Or a pandemic, hurricane, trade war, earthquake, alien invasion, and so on could delay the shipment of your new servers. (Not that anything as

far-fetched as a pandemic might occur.) Or one of your developers might move to Iceland. How can you handle these sorts of completely unpredictable problems?

It would be essential to always have a contingency plan in place to prepare for the worst-case scenario at all times. Some of the top priorities to have to help prevent these kinds of unpredictable situations would be to have a risk management plan in place. This would assess potential risks and have preparations in place to support ongoing projects in case they may have to be placed on hold due to a global pandemic for example. Additionally, having support staff available like contracted developers on standby in case a core developer is out sick and there is a deadline to meet. This would really have to be a system already in place of developers familiar with the product because otherwise, they would have to be trained on the product before supporting the project's completion and delivery. Lastly, to always expect the unexpected and prepare for the worst always.

4.8 [Stephens page 79]

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

1. Ignoring the problem of a task going behind schedule and hoping you can make up the time later. Unless you know for a fact that you have the time to make up the task later, assume you don't and that you'll fall farther behind.
2. Pile extra developers on the task and assume they'll reduce the time needed to finish it. It takes time for new people to get up to speed on any task, so they may instead increase the overall time it takes to finish the task.

5.1 [Stephens page 114]

List five characteristics of good requirements.

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

5.3 [Stephens page 114]

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 application requirements.

- *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.*
- *c. Let the user specify upload/download parameters such a number of retries if there's a problem.*
- *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.*
- *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 whan another is in progress, it waits until the other one finishes.*
- *j. Perform schedule uploads/downloads.*
- *k. Keep a log of all attempted uploads/downloads and whether the succeeded.*
- *l. Let the user empty the log.*
- *m. Display reports of upoad/download attempts.*
- *n. Let the user view the log reports on a remote device such as a phone.*
- *o. Send an e-mail 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 it's maximum retury umber of times.*

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

- <u>Business Requirements:</u>
  - m. Display reports of upload/download attempts.
- <u>User Requirements:</u>
  - 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.
  - c. Let the user specify upload/download parameters such as the number of retries if there's a problem.
  - 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.
  - f. Allow uploads/downloads to run 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.
- <u>Functional Requirements:</u>
  - 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.
  - j. Perform scheduled uploads/downloads.
  - k. Keep a log of all attempted uploads/downloads and whether they succeeded.
- <u>Non-Functional Requirements:</u>
  - o. Send an e-mail 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.

5.9 [Stephens 115]

Figure 5-1 [right] 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, "Contratulations, 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.

1. <u>Must-haves:</u>
    a. Basic Functionality: Ensure that the core functionality of the hangman game remains intact, including the ability to start a new game, guess letters, and display the outcome.
    b. Random Word Generation: Continue using a large list for random word selection to maintain the game's variability.
2. <u>Should-haves:</u>
    a. Enhanced User Interface (UI): Improve the UI for a better user experience, including more visually appealing graphics and animations for Mr. Bones's skeleton.
    b. Word Categories: Introduce word categories (e.g., animals, fruits, countries) for a more thematic experience.
3. <u>Could-haves:</u>
    a. Multiplayer Mode: Add a multiplayer option, allowing users to play against friends or other online players.
    b. Difficulty Levels: Implement different difficulty levels, with easier words for beginners and more challenging words for advanced players.
    c. Hints System: Include a hint system to assist players who may be struggling with a particular word.
4. <u>Won't-haves:</u>
    a. Complex Storyline: Avoid adding a complex storyline to keep the game simple and accessible.
    b. In-app Purchases: Decide not to include in-app purchases to maintain a free and inclusive experience.