Dylan Suzuki, Jackson Muller, Aidan Hodges, Isaiah Pajarillo
BJ Johnson
CMSI 4072
2/4/26

Homework 1

Problem 1.1, Stephens page 12

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

- Step 1: Requirement Gathering
- Step 2: High Level Design
- Step 3: Low Level Design
- Step 4: Development
- Step 5: Testing
- Step 6: Deployment
- Step 7: Maintenance
- Step 8: Wrap Up

Problem 1.2, Stephens page 12

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

- Step 1: Requirement Gathering
  - Requirement gathering is the process of polling customers to generate a list of things that the software needs to have/do.
- Step 2: High Level Design
  - Once you have some semblance of the scope and requirements of the project, the high level design is where you figure out the overall structure of the project and how, in broad strokes, the project will accomplish its tasks.
- Step 3: Low Level Design
  - Once you know what the project should do from a bird's-eye view, the low level design is where you figure out the weeds of how the project will actually do the things that the high level design asks it to.
- Step 4: Development

- Once the design (or a version of it) is finalized, development is where programmers actually build the software described by the low level design.

- Step 5: Testing
  - While testing should be done throughout development, there should also be some kind of formal testing done by testers who know what to look for.
- Step 6: Deployment
  - Once the software is sufficiently bug-free, deployment is the process of taking your software and getting it set up where it needs to be, which can involve a lot of time consuming tasks.
- Step 7: Maintenance
  - After the software has been released into the wild, users will inevitably find bugs or improvements that need to be addressed, which will happen during the maintenance stage.
- Step 8: Wrap Up
  - After releasing the software, it's important to debrief in order to take note of things that went well, went horrifically wrong, or could be improved a bit.

Problem 2.4, Stephens page 26

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.

Investigation: Compare this process to what you can do with GitHub versions. How are the two tools different? How are they the same?

- These are changes

- Docs and Github are clearly designed for vastly different purposes. While the version history page of Docs does give a "git diff" style overview of changes, the two services differ greatly in their everyday use. As someone who once tried using the VSCode extension that allows for Google Docs style editing, having multiple people editing a single document in real time is TERRIBLE for programming. Additionally, Github has extensive support for branching, tagging, merging, etc. to allow for the development of individual modules. In contrast, Google Docs is more concerned with keeping track of a single main document with a unified history.

Problem 2.5, Stephens page 27

What does JBGE stand for and what does it mean?

- JBGE means "Just Barely Good Enough," and refers to the mentality that documentation should just barely be good enough, lest you waste precious programming time on documentation. The problem with this is that many JBGE approaches aren't actually good enough, and leave development teams struggling to understand their codebases months or years down the line.

Data for Problems 4.2 and 4.4

Table 4.2 [below] summarizes some of the classes and modules you might need (and their unreasonably optimistic expected times) to develop players and zombies for the game. (The program would also need lots of other pieces not listed here to handle other parts of the game.)

Use the following table of data for Exercises 4.2 and 4.4.

Task	Time (Days)	Predecessors
A. Robotic control module	5	—
B. Texture library	5	C
C. Texture editor	4	—

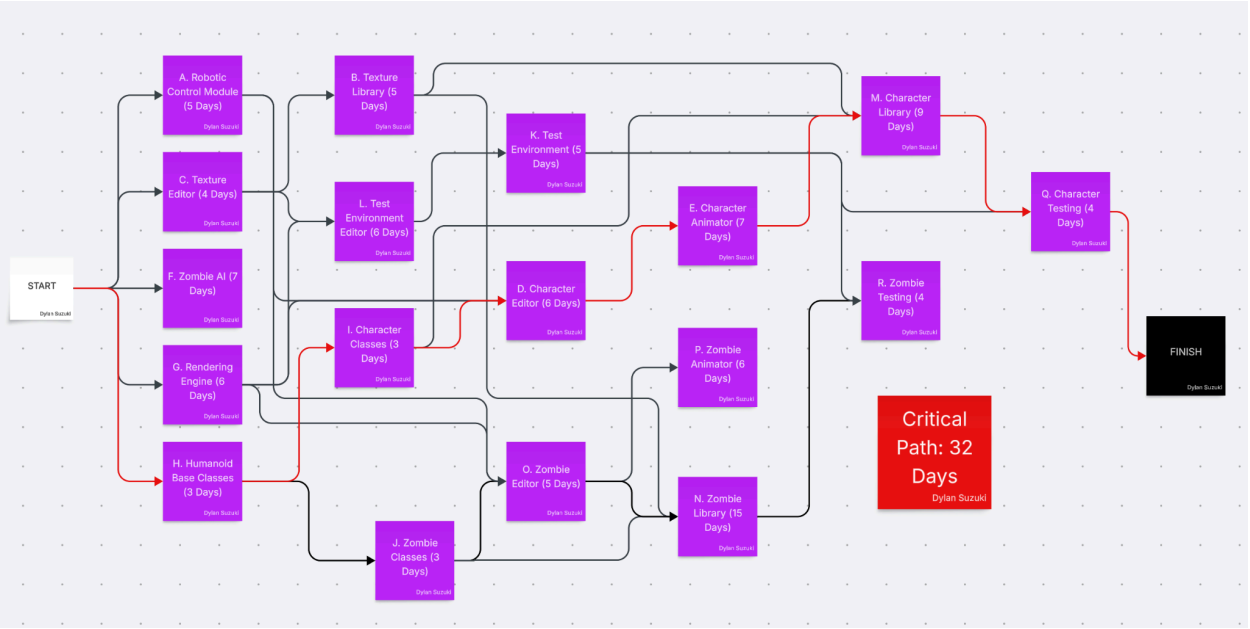
D. Character editor	6	A, G, I
E. Character animator	7	D
F. Artificial intelligence (for zombies)	7	—
G. Rendering engine	6	—
H. Humanoid base classes	3	—
I. Character classes	3	H
J. Zombie classes	3	H
K. Test environment	5	L
L. Test environment editor	6	C, G
M. Character library	9	B, E, I
N. Zombie library	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

Problem 4.2, Stephens page 78

1. Use `critical path methods` to find the total expected time from the project's start for each task's completion.

Task	Time	Predecessors	Expected Time
A. Robotic Control Module	5		5
B. Texture Library	5	C	9
C. Texture Editor	4		4
D. Character Editor	6	A, G, I	12
E. Charactor Animator	7	D	19
F. AI (Zombie)	7		7
G. Rendering Engine	6		6
H. Humanoid Base Classes	3		3
I. Character Classes	3	H	6
J. Zombie Classes	3	H	6
K. Test Environment	5	L	17
L. Test Environment Editor	6	C, G	12
M. Character Library	9	B, E, I	28
N. Zombie Library	15	B, J, O	26
O. Zombie Editor	5	A, G, J	11
P. Zombie Animator	6	O	17
Q. Character Testing	4	K, M	32
R. Zombie Testing	4	K, N	30

2. Find the critical path. What are the tasks on the critical path?



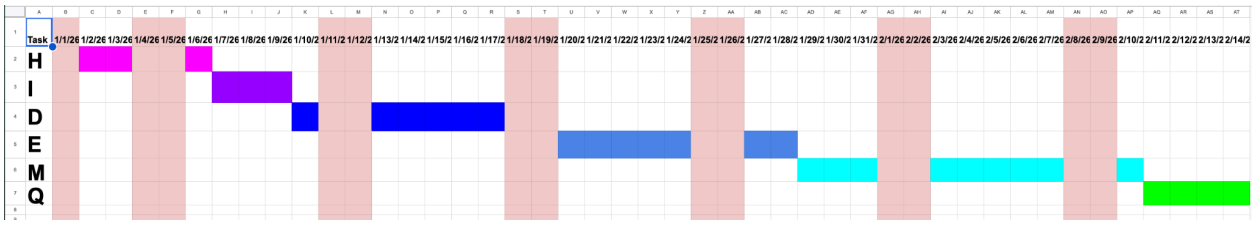
3. What is the total expected duration of the project in working days?
    a. The total expected duration of the project is 32 days

Problem 4.4, Stephens page 78

Build a Gantt chart for the critical path you drew in Exercise 2. 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
St. Valentine's Day	February 14
Alien Overlord Appreciation Day	March 26

Income Tax Day April 15



Problem 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, like 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, right?] Or one of your developers might move to Iceland, which is a real nice place to raise your kids up. How can you handle these sorts of completely unpredictable problems?

- The best way to handle these sorts of problems is to re-evaluate the schedule as soon as possible. If the "sick time" tasks don't account enough for the margin you need, you'll have to adapt as soon as possible so everyone is on the same page. Another way the text suggests dealing with these big decisions is having the management do extensive risk management in order to determine which changes are worth it/possible.

Problem 4.8, Stephens page 79

According to your textbook, what are the two biggest mistakes you can make while tracking tasks?

- The text says that the first biggest mistake is ignoring when developers start to fall behind schedule in hopes that time will be made up later. Rather than push the developer to stick to the schedule adamantly, the text says managers will get much

better results from getting more accurate estimates from the developers or estimates with some level of margin. The second biggest mistake according to the text is throwing more developers at the problem. Once a project is sufficiently complex, adding more developers will only add more time to the development since those new people need to be brought up to speed before they can become useful.

Problem 5.1, Stephens page 114

Operating Systems security, Application security, Data Security, Network Security Physical Security

Problem 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 applications requirements.

- a. Allow users to monitor uploads/downloads while away from the office.

  User, Functional

- b. Let the user specify website log-in parameters such as an Internet address, a port, a username, and a password.

  User, Functional

- c. Let the user specify upload/download parameters such a number of retries if there's a problem.

  User, NonFunctional

- d. Let the user select an Internet location, a local file, and a time to perform the upload/download.

  User, Functional

- e. Let the user schedule uploads/downloads at any time.

  User, Nonfunctional, sort of Functional because it is still a function of the app

- f. Allow uploads/downloads to run at any time.

  Nonfunctional

- g. Make uploads/downloads transfer at least 8 Mbps.

  Nonfunctional

- h. Run uploads/downloads sequentially. Two cannot run at the same time.

  Nonfunctional

- i. If an upload/download is scheduled for a time whan another is in progress, it waits until the other one finishes.

  Nonfunctional

- j. Perform schedule uploads/downloads.

  Functional

- k. Keep a log of all attempted uploads/downloads and whether the succeeded.

  Functional

- l. Let the user empty the log.

  User, Functional

- m. Display reports of upload/download attempts.

  Functional

- n. Let the user view the log reports on a remote device such as a phone.

  User, Functional

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

  Functional and Nonfunctional

- p. Send a text message to an administrator if an upload/download fails more than it's maximum return number of times.

  Functional and Nonfunctional

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

^^^, there are requirements in every category

Problem 4.9, Stephens page 83-84

Figure 4-1 [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, "Congratulations, you won!" If you build Mr. Bones's complete skeleton, a message says, "Sorry, you lost."

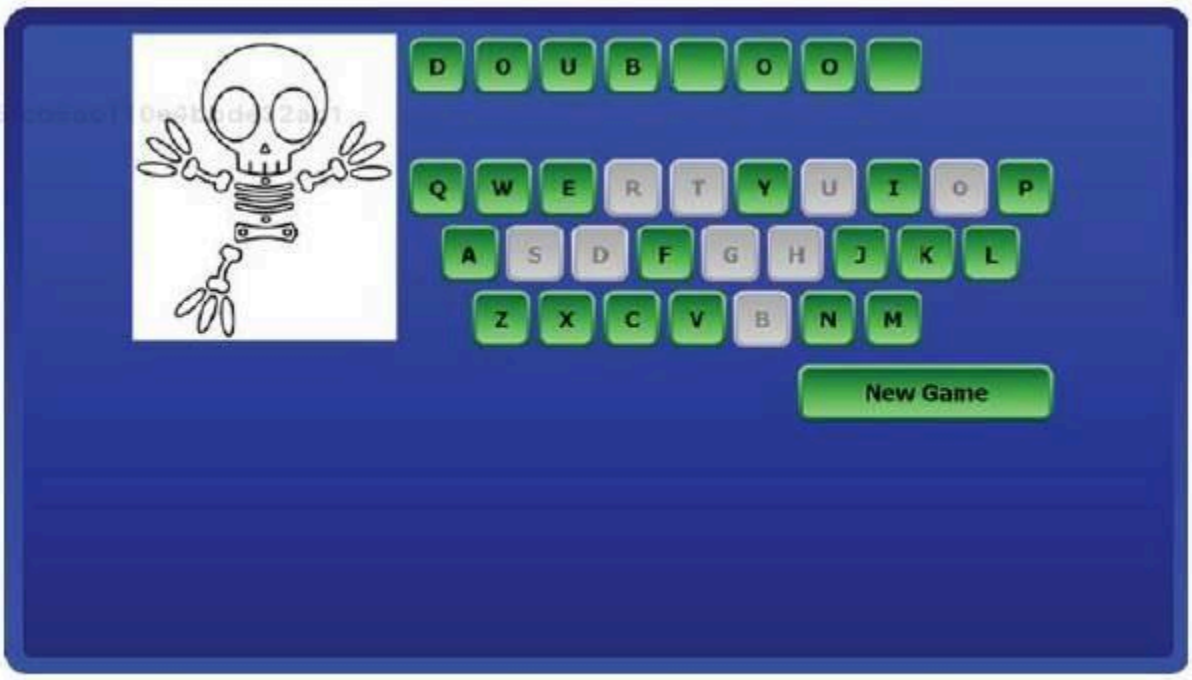


FIGURE 4-1: The Mr. Bones application is a hangman word game for Windows Phone.

Brainstorm this application and see if you can think of ways you might change it. Use the MOSCOW method to prioritize your changes.

- **Must have:** New game button, game loop (Typing/guessing script, win/lose condition, correct or non-correct function), readable UI

- **Should Have:** A unique layout, New game button, Mr. Bones, accessibility options such as sound or alternate color schemes.
- **Could Have:** An “already guessed” feature, a more cleaner/more satisfying layout (i.e cleaner colors and a lack of overly blank space at the bottom). A better design for Mr.Bones instead of having him in an awkward white box that doesn't fit the rest of the screen.
- **Won't Have:** Microtransactions, Paid subscription