Allen Vartanian

Dr. Johnson

CMSI 402-01

29 January 2018

Homework 1

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

- The basic tasks are requirements gathering, high-level design, low-level design, development, testing, deployment, maintenance, and wrap-up.

1.2) Give a one sentence description of each of the tasks you listed for Exercise 1.

- **Requirements Gathering:** Finding out what the customers want and what the customers need.
- **High-Level Design:** Includes such things as decisions about what platform to use (such as desktop, laptop, tablet, or phone), what data design to use (such as direct access, 2-tier, or 3-tier) and interfaces with other systems (such as external purchasing systems).
- **Low-Level Design:** Includes information about *how* that piece of the project should work.
- **Development:** The programmers write the code, testing as they go to find and remove as many bugs as they reasonably can.
- **Testing:** First developers test their own code. Then testers who didn't write the code test it. After a piece of code seems to work properly, it is integrated into the rest of the project, and the whole thing is tested to see if the new code broke anything.

- **Deployment:** You roll out your software, and ideally the users are overjoyed, and everyone lives happily ever after.
- **Maintenance:** As soon as the users start pounding away at your software, they'll find bugs and think up a slew of enhancements, improvements, and new features that they want added immediately.
- **Wrap-Up:** Need to perform a post-mortem. You need to evaluate the project and decide what went right and what went wrong.

2.4) Like Microsoft Word, Google Docs provides some simple change tracking tools. Go to http://www.google.com/docs/about/ to learn more and to sign up. Then create a document, save it, close it, reopen it, and make changes to it as you did in Exercise 2.

- Ok, I did it.
- Google Docs shows your version history on a sidebar. Due to Google Docs auto-saving after every pause in progress, there are quite few versions, so it collapses versions made in separate sessions into individual dropdowns of each session, roughly.

2.5) What does JBGE stand for and what does it mean?

- It means "just barely good enough," with regards to documentation and comments. The idea is that if you provide too much documentation, you end up wasting a lot of time updating it as you make changes to the code.

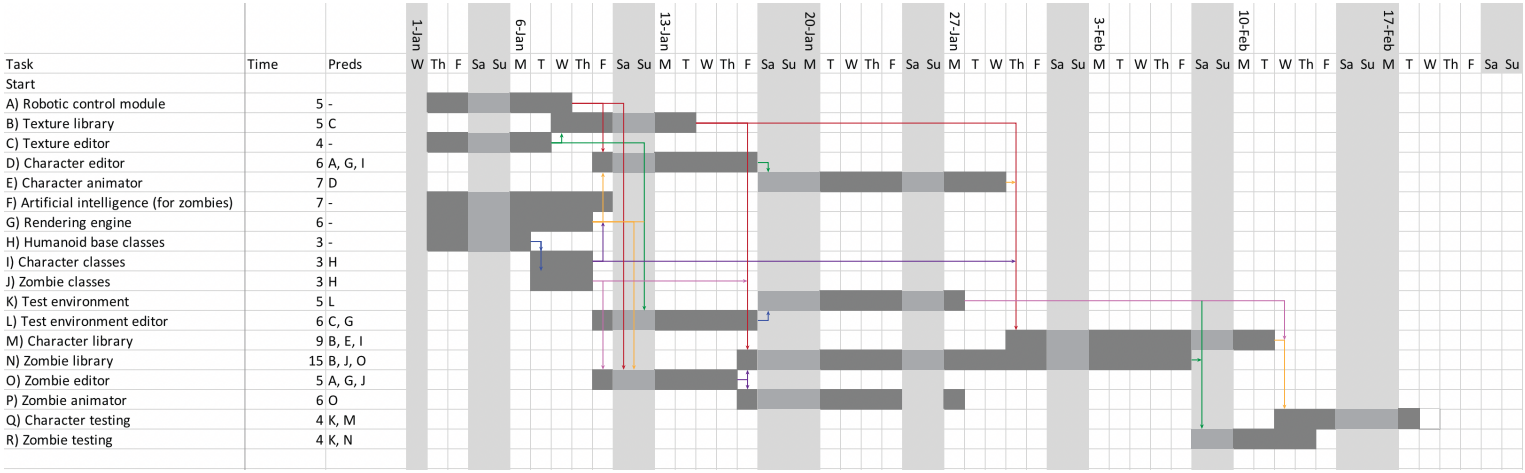
3.2) Use critical path methods to find the total expected time from the project's start for each task's completion. Find the critical path. What are the tasks on the critical path? What is the total expected duration of the project in working days?

- Assuming a start state exists: Start(0) > H(3) > I(6) > D(12) > E(19) > M(28) > Q(32)

- Start() > H(Humanoid base classes) > I(Character classes) > D(Character editor) > E(Character animator) > M(Character library) > Q(Character testing) Total Time: 32 days

3.4) Build a Gantt chart for the network you drew in Exercise 3. Start on Wednesday, January 1, 2020, 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



3.6) 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 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 task's time estimate by some amount.
  - Usually time just ends up being used up for no good reason and total project time gets extended
- Add specific tasks to the project to represent lost times.
  - Add tasks to include vacation and "sick" time
- Plan for approvals, order lead times, and setup
  - Might as well make them tasks in your schedule

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

- Ignoring the problem and hope you can make up the time later.
- Piling on extra developers on the task and assume they can reduce the time needed to finish it.

4.1) List five characteristics of good requirements.

- Clear, unambiguous, consistent, prioritized, and verifiable.

4.3) 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.

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 when 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 they succeeded.

l. Let the user empty the log.

m. Display reports of upload/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 its maximum retry number of times.

For this exercise, list the audience-oriented categories for each requirement. Are there requirements in each category?

- **a)** User requirements
- **b)** User requirements
- **c)** User requirements
- **d)** User requirements
- **e)** User requirements

- **f)** Functional requirements
- **g)** Nonfunctional requirements
- **h)** Functional requirements
- **i)** Functional requirements
- **j)** Functional requirements
- **k)** Functional requirements
- **l)** User requirements
- **m)** Functional requirements
- **n)** User requirements
- **o)** Functional requirements
- **p)** Functional requirements
- No implementation or business requirements

4.9) 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' 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' complete skeleton, a message says, "Sorry, you lost."

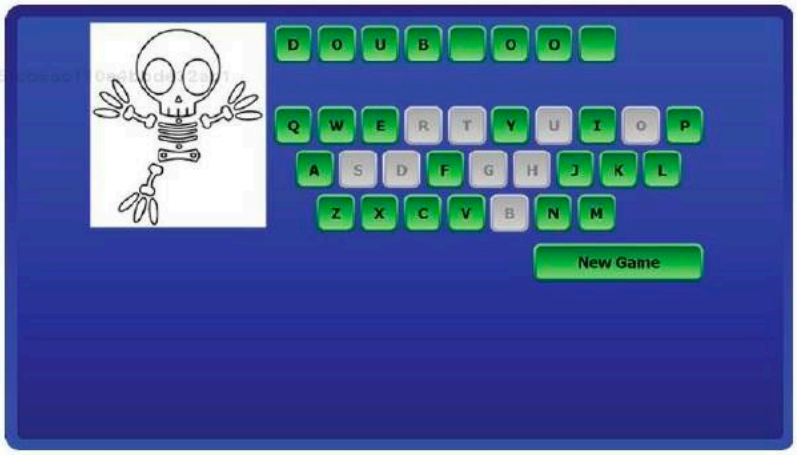


FIGURE 4-1: The Mr. Bones application is a hangman word game for Windows Phone.

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

- **S)** Keep a score based on how many letters it took the player to guess the word (i.e. 100pts for only guessing the letters in the word and subtract 5 points for every wrong letter).
- **S)** Keep highest score in memory so player is encouraged to keep playing and beat score.
- **S)** Display the word upon loss for the player to learn for future reference
- **C)** Let the user guess the whole word at a time
- **C)** Have skill levels for various players
- **C)** Have a “Quit Game” button to exit the game
- **C)** Have an undo button for beginner players