Geoffrey Colman

CMSI 402

6 February 2019

**HW #1**

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

Requirements gathering, high- and low-level design, testing, development, deployment, wrap-up, and maintenance.

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

*Requirements gathering* is the process of determining what the customer/user can expect of the “finished” software product in terms of functionality, and what the development team is responsible for building.

*High-level design* provides a broad, architectural overview of what all the major functional “chunks” of a software project (e.g., database backend, user interface, custom data objects) are expected to be.

*Low-level design* provides details necessary to the actual implementation of each of the major functional pieces.  
*Development* involves all of the work actually writing code and building the software.

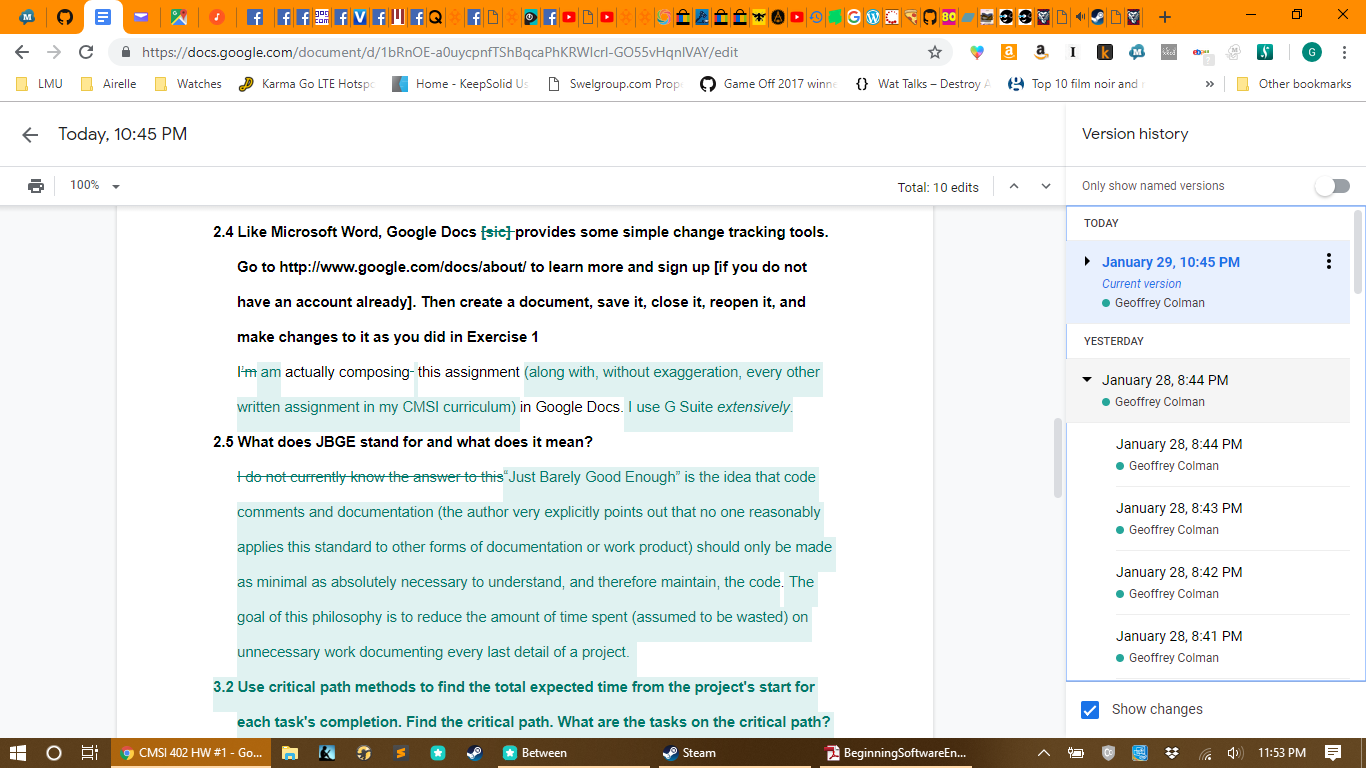
*Testing* demonstrates the presence of bugs or errors.

*Deployment* is the process by which the software is published/released/otherwise put in place for users.

*Maintenance* is the (ongoing) process by which future bugs are fixed and additional features are added after the initial release.

*Wrap-up* is the “post-mortem” process in which a team determines what parts of the development process went well, what parts did not, and how to both reinforce the positive aspects and avoid the negative aspects.

**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 sign up [if you do not have an account already]. Then create a document, save it, close it, reopen it, and make changes to it as you did in Exercise 1**

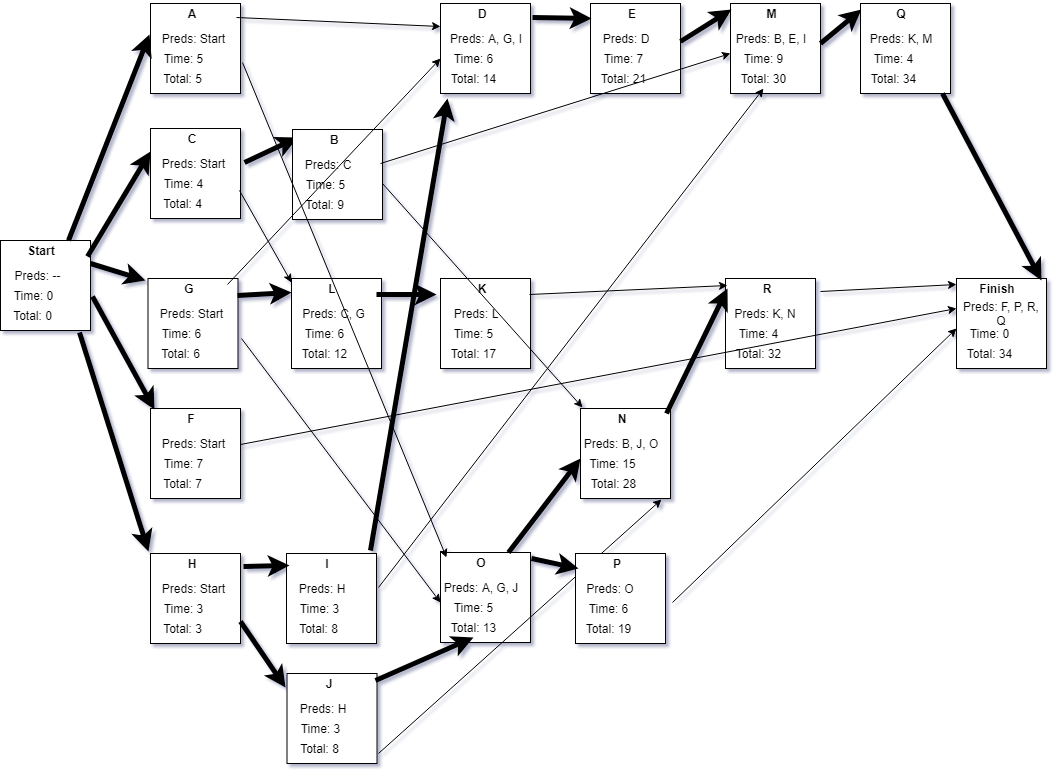
I am actually composing this assignment (along with, without exaggeration, every other written assignment in my CMSI curriculum) in Google Docs. I use G Suite *extensively*. 

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

“Just Barely Good Enough” is the idea that code comments and documentation (the author very explicitly points out that no one reasonably applies this standard to other forms of documentation or work product) should only be made as minimal as absolutely necessary to understand, and therefore maintain, the code. The goal of this philosophy is to reduce the amount of time spent (assumed to be wasted) on unnecessary work documenting every last detail of a project.

**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?**

As it was easier for me to follow the author’s algorithm by actually building the PERT network, I did so (also “GC Exercise 3.2.png”):

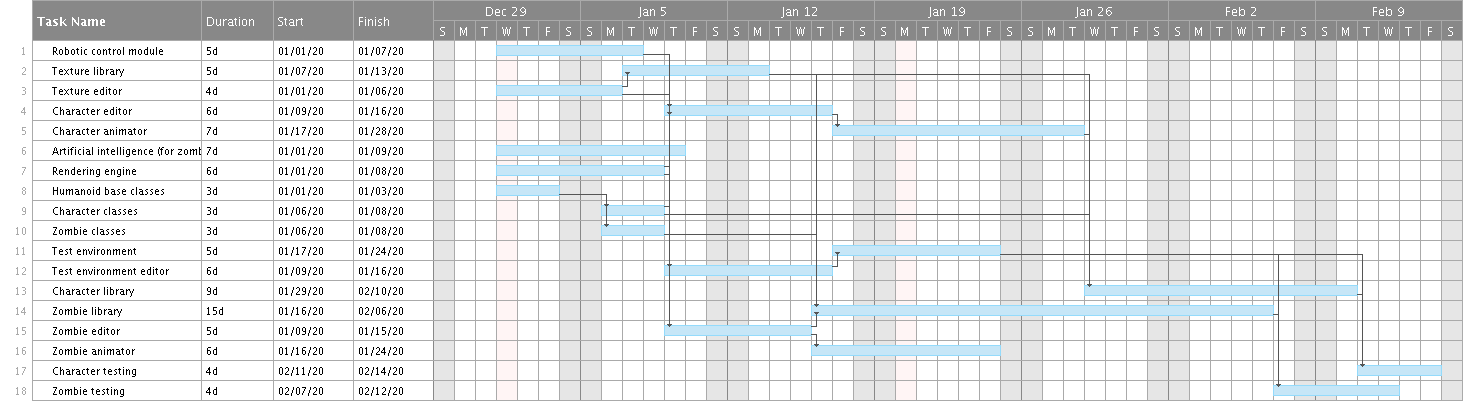


Thus, the critical path of 34 working days follows the following task flow:

“Start→H→I→D→E→M→Q→Finish”

For the sake of brevity, refer to the original table for full task descriptions associated with each letter.

**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 [New Year’s Day, MLK Day, or President’s Day].**



Vis-à-vis the question posed by the book exercise: we would expect to complete the project on Friday, the 14th of February (awww, Valentine’s Day) 2020.

(If this image is too difficult to read/view here, refer to “GC Exercise 3.4.png”)

**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?**

One way to deal with these sorts of unforseen problems is to build them right into the project’s task estimates from the beginning: add explicit tasks (the author says “at the end,” and then allocate their time as such incidents come up) to the project to provide for the possibility of sick leave, vacation time, and frictional logistic delays that can reasonably be assumed to crop up in any project.

Another risk management strategy that can be applied to these sorts of problems is the development of “workarounds” for a task that falls woefully behind schedule. Being able to provide an alternative solution (or sometimes even bump the task to the next release if it isn’t determined to be “mission-critical” with respect to requirements and dependencies) can help to mitigate such delays, even if it’s less than ideal. The author provides the example of purchasing office supplies by directly contacting the purchasing agent as a workaround to utilizing an automated system: it’s vastly simpler to implement, even if it’s far less efficient in the long-term.

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

Certainly the biggest mistake is to let any potential snags or delays go *completely* unaddressed. One can’t simply ignore a problem and hope all will be well in the end (in this specific case, assuming that developers will be able to make up lost time), but rather should assume that the delays will actually worsen as they propagate.

The other major mistake is the age-old (one might even say “time-honored”) “solution” of proverbially throwing more people at the problem. As discussed in the inimitable *The Mythical Man-Month*, this often has the counterintuitive effect of actually slowing development down as new team members acclimate to the team and familiarize themselves with a new codebase (as well as potentially any number of new tools and technologies).

**4.1 List five characteristics of good requirements.**

*Clear* (well-defined and easy to understand), *unambiguous* (only one way to interpret them), *consistent* (they contradict neither each other nor themselves, and allow sufficient freedom for the problem to remain solvable), *prioritized* (ordered on the basis of importance), and *verifiable* (limited and precisely defined in order to determine if the application meets them).

**4.3 Suppose you want to build a program called TimeShifter to upload and download files at scheduled times while you're on vacation. For this exercise, list the audience-oriented categories for each requirement. Are there requirements in each category? If not, state why not.**

**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 as number of retries if there's a problem.** *- User, Functional*

**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, Functional*

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

**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 when another is in progress, it waits until the other one finishes.** *- Nonfunctional*

**j. Perform scheduled uploads/downloads.** *- Functional*

**k. Keep a log of all attempted uploads/downloads and whether they 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*

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

There are no requirements in the business or implementation categories. I can’t really speculate as to the specific motivations of the person or group who created this list of requirements for this application, but - regarding the former - the customer likely did not need to specify any overarching business goals that they hoped the software would help them accomplish. The author asserts that these likely *should* be rolled into a business case document, anyway. The latter is likely due in large part to the nature of the application being a fresh start, and not a substantial transition from an existing system, but could also involve an oversight - on the part of both the development team and the customer - for not specifying details about the manner in which the application will be published/distributed, installed, etc. For example, the author specifically cites training materials (e.g., setup documentation, help files) and training users to use the system as examples of an implementation requirement. Anything having to do with post-completion rollout was evidently neglected in this list of requirements.

**4.9 Figure 4-1 shows the design for a simple hangman game that will run on smartphones. Brainstorm this application and see if you can think of ways you might change it. Use the MOSCOW method to prioritize your changes.**

The first, most obvious change that popped into my head is some form of “title” or home screen, potentially allowing the user to select options (such as difficulty, font/size, color scheme, etc.) before beginning a game, and potentially different game modes that could be developed in the future. Using MOSCOW, I would prioritize this is as a “should” - it adds considerable user value (how many applications have you used that don’t have any form of settings menu? I would wonder why it didn’t...) but doesn’t seem *strictly* *necessary* in order to have a minimum viable product.

Following on from this, some of those options would form quality-of-life improvements for users that I would prioritize as “could” - it would be nice if the game supported alternative themes/color schemes for users with green/blue color blindness, for example, but this likely wouldn’t be a top priority for the initial release. Other examples could include support for additional languages (utilizing external dictionaries as parameters seems like it would make this extremely easy to abstract, but it would be reasonable to assume that the team would prefer to have a working version of the software support a single language before expanding to others), adjustable difficulty level (which could be tricky to implement in a meaningful fashion, as the number of letters in the word could be misleading for words such as “Mississippi” that include repetitions of common letters), and customizable text (simple in theory, but could easily break the carefully architected layout if not comprehensively tested).

The option for some sort of competitive/multiplayer game mode would *significantly* enhance the product for the user, and is thus a “should.” I can imagine most people nowadays would likely consider this a “must,” but I beg to differ: a casual mobile phone game doesn’t *absolutely require* competitive online (or offline, if using the low-tech, turn-based approach on a single device employed during the ‘80s and ‘90s) play in order to be a viable product. Various features to support multiplayer (such as a global leaderboard, direct PvP online play) could shake out as “should,” “could,” or “won’t” depending upon the specific design path chosen. To this end, however, some unambiguous means of “scoring” is a “must” *within* the context of multiplayer.

I genuinely can’t think of anything I would label a “must” that wasn’t addressed by the initial description, but I freely admit that I have low expectations for a game described as “simple hangman.” Similarly, I can’t think of any changes that I would prioritize as a “won’t,” as anything that I can think of that would fall under that category is not a change I would seriously propose.