Carleen Petrosian
CMSI 402
01-24-2018

Assignment #1

1. The basic tasks that all software engineering projects must handle are: requirements gathering, high-level design, low-level design, development, testing, deployment, maintenance, wrap up.

2.
Requirements gathering: finding out what the customer wants and needs and documenting them into requirements specifications; these documents can tell the customer what they will be getting and the project members what they will be building; members can refer to these specifications in the duration of the project to make sure it's heading in the right direction

High-level design: decisions on what platform to use (such as desktop, laptop, tablet, or phone), what data design to use (such as direct access, 2-tier, 3-tier), and interfaces with other systems (such as external purchasing systems); also include information about the project architecture at a high level; high level design should cover every aspect of the requirements

Low-level design: after the high-level design breaks the project into pieces, the low-level design contains information on how those pieces of the project should work; they should give enough guidance to the developers who will implement those pieces

Development: continue refining the low-level designs until you are able to implement those design into code; find and remove any bugs as you can while you are writing and testing the code

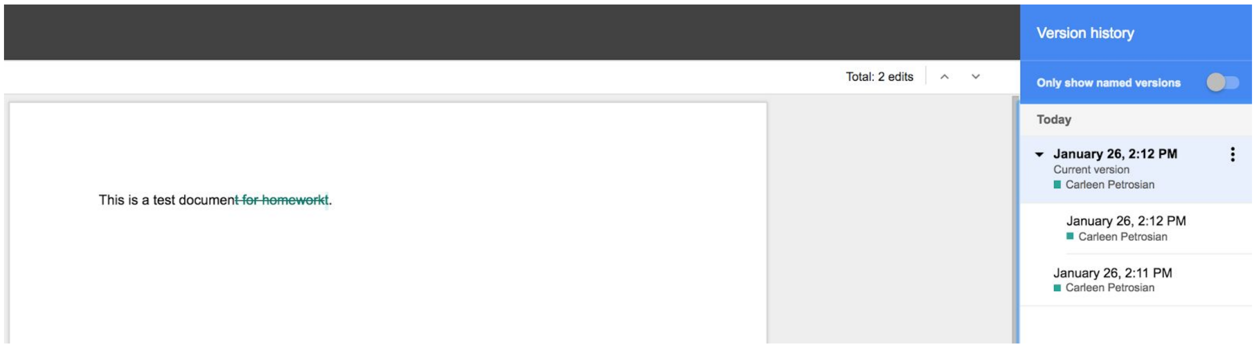
Testing: developers should test their own code, then give it to someone who didn't write it to test it; when that piece seems to be working properly, integrate it into the rest of the project and test the whole thing to make sure nothing was broken; when tests fail, go back into the code to find and figure out the bug

Deployment: essentially releasing the project to the customer for use; can be time consuming, difficult, and expensive; may include other steps to take to make sure that the user is able to use the product, there aren't any difficulties, identifying new bugs to fix, etc.

Maintenance: as users continue to use the product, the more bugs they might find that needs fixing; if they enjoy the product, they'll be asking for enhancements, improvements, new features, etc. that you can then implement

Wrap-up: evaluate the project and identify what went right and what went wrong; figure out how to make things that went well happen more in the future and find ways to prevent things that went badly in the future

3.



This is a test document~~ for homeworkt~~.

January 26, 2:12 PM
Current version
Carleen Petrosian

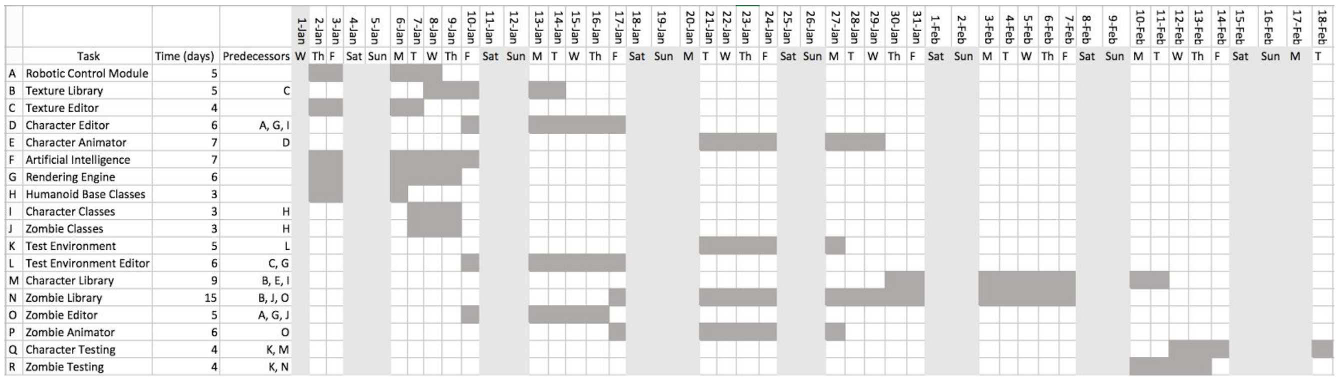
January 26, 2:12 PM
Carleen Petrosian

January 26, 2:11 PM
Carleen Petrosian

4. JBGE stands for "just barely good enough." In code 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. This philosophy can reduce the amount of documentation you produce, without eliminating them entirely. It other words, you shouldn't write any more code documentation or comments than absolutely necessary.

5. The critical path runs through tasks G, D, E, M, and Q. The total expected duration of the project is 32 working days.

6.



				1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan	8-Jan	9-Jan	10-Jan	11-Jan	12-Jan	13-Jan	14-Jan	15-Jan	16-Jan	17-Jan	18-Jan	19-Jan	20-Jan	21-Jan	22-Jan	23-Jan	24-Jan	25-Jan	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	31-Jan	1-Feb	2-Feb	3-Feb	4-Feb	5-Feb	6-Feb	7-Feb	8-Feb	9-Feb	10-Feb	11-Feb	12-Feb	13-Feb	14-Feb	15-Feb	16-Feb	17-Feb	18-Feb
	Task	Time (days)	Predecessors	W	Th	F	Sat	Sun	M	T	W	Th	F	Sat	Sun	M	T	W	Th	F	Sat	Sun	M	T	W	Th	F	Sat	Sun	M	T	W	Th	F	Sat	Sun	M	T	W	Th	F	Sat	Sun	M	T	W	Th	F	Sat	Sun	M	T
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	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																																										▨	▨	▨	▨					

7. You can add tasks at the end of the schedule to account for completely unexpected problems. When one of these problems does occur, you can insert its lost time into the schedule.

8. One of the biggest mistakes you can make while tracking tasks is ignoring the problem and hoping to make the time up later. Unless you have a solid reason to believe that you can catch

up, you must assume that you'll fall further behind. The second biggest mistake you can make is to pile extra developers on a task and assume that they can reduce the time to finish it. Although adding people on a task can be beneficial at times, most of time it can cause the task to take longer, since it can take time for people to get up to speed on the task.

9. Good requirements are clear, unambiguous, consistent, prioritized, and verifiable.

10.
(a) business requirements
(b) user requirements or functional requirements
(c) user requirements or functional requirements
(d) user requirements or functional requirements
(e) non functional requirements
(f) non functional requirements
(g) non functional requirements
(h) non functional requirements
(i) non functional requirements
(j) functional requirements
(k) functional requirements
(l) user requirements or functional requirements
(m) user requirements or functional requirements
(n) user requirements or functional requirements
(o) user requirements or functional requirements
(p) user requirements or functional requirements

All of the audience-oriented categories are present in these requirements, except for the implementation requirements. You might eventually need to purchase some hardware later to support the application, but so far the program is just uploading and downloading files, which doesn't require any implementation requirements.

11.
(S): the program should calculate a score
(S): if the program calculates a score, it should keep track of those scores so the user can beat their previous high score
(C): the program can allow the ability for the user to guess the whole word for extra points
(C): the program can have multiple skill levels, where the difficulties of the words would increase as the skill level increased