Ian Lizarda

Prof. Johnson

CMSI-401

10/30/2019

Assignment 2 Questions

- 1) Write a short paragraph to answer these three questions:
    - What are the two major concerns of any software project?
        - How much will it cost?
        - How long will it take?
    - Which do you feel is more important?
        - I think cost is more important. In regards to many applications that have been pushed out, most have some leeway in terms of deadlines. If trajectory cannot be reached, more time can be allotted. However, if there is not enough funding then the app will most likely fail because an incomplete app with no infrastructure is almost always worse than a delayed app.
    - Where does the idea of complete functionality fit with these two concerns?
        - The idea of complete functionality means the usually unattainable standard that both cost and time are on/under budget and on/under time, with everything that is needed & works great.
- 2) In the Agile method for software development, what are the four main phases that occur in each and every iteration? Do you feel that any of them could be done at the start of the project and not be repeated in every iteration? Do you feel that would save time overall on the project? Justify your answers with a brief explanation.
    a) The four main phases are requirements, design, code, and test. I feel that the requirements should be not repeated for every iteration because that will lead to code that is constantly changing if the requirements change every time. The requirements should be laid out at the beginning of the project with minimal changes. I feel that would save time because the developers **know** what they need to build, and don't have to worry so much about any impending changes. It would also save cost, as requirements that change usually imply that the cost will increase.
- 3) In the Waterfall method for software development, what are the main phases that occur? How are they different from the phases in the Agile method? What other phases are in Waterfall that are left out of Agile? Do you think these are needed in Waterfall? Describe a situation using Agile in which one of these extra Waterfall phases might me needed.
    a) The main phases of the waterfall method of software development are requirements analysis, design, code, test, and maintenance. Other than the maintenance phase, it is virtually identically to agile. The reason for this phase is because the waterfall method is one giant sprint with room for maintenance only at the end. Agile could benefit from having a maintenance phase because the

sprint aspect of agile means that bugs can get through each sprint. Having maybe a dedicated sprint at the end for maintenance could be super beneficial.

- 4) Write one-sentence answers to the following questions:
    - What is a “user story”?
        - A user story is essentially the description from the customer’s POV of how they interpret and imagine interacting with the software that is being produced for them.
    - What is “blueskying”?
        - “Blueskying” is a brainstorming session between the customer and contractor. The goal is for the customer’s ideas to be documented so that all of the customer’s needs are described and understood by everyone.
    - What are four things that user stories SHOULD do?
        - They should describe the one thing that the customer needs the software to do
        - They should be written in the language that the customer understands
        - They should be concise
        - They should be written by the customer
    - What are three things that user stories SHOULD NOT do?
        - They should **NOT** be long
        - They should **NOT** contain technical jargon that the customer might not understand
        - They should **NOT** mention any specific technologies
- 5) What is your opinion on the following statements, and why do you feel that way:
    - All assumptions are bad, and no assumption is a good assumption.
        - I semi-agree. I do think assumptions are bad, but they should not be made in the first place. It should always be a goal to clarify and understand what the customer wants. If assumptions are to be made, it should be done with good judgement and good reason. Other than that, proper clarification and communication should be the goal.
    - A big user story estimate is a bad user story estimate.
        - I totally agree. There is a balance in regards to how big a user story should be, but if the big picture ends up being too big, the application as a whole can suffer. It is better to start off small and iteratively add features, as opposed to trying to fit in all the features at once. If the scope is too big, there will not be enough scalability and maintenance for the application to be feasible. There would probably be a lot of bugs, and more money and time would get poured into the product than needed. With that extra time, priorities can switch and thus lead to lots of conflict. User stories should max be 15 days, so you can change if any problems happen.

- 6) Fill in the blanks in the statements below, using the following things [you can use each thing for more than one statement]: Blueskying; Role playing; Observation; User story; Estimate; Planning poker.
    - You can dress me up as a use case for a formal occasion: User Story
    - The more of me there are, the clearer things become: User Story
    - I help you capture everything: Blueskying, Observation
    - I help you get more from the customer: Blueskying, Role Playing, Observation
    - In court, I'd be admissible as firsthand evidence: Observation
    - Some people say I'm arrogant, but really I'm just about confidence: Estimate
    - Everyone's involved when it comes to me: Blueskying
    - NOTE: when you have finished, check your answers with the result in your text on page 62. Do you agree with the book answers? If you disagree with any of them, justify your preferred answer.
        - I think that blueskying fits the "I help you get more from the customer" case because you can just ask for ideas from the customer.

- 7) Explain what is meant by a better than best-case estimate.
    - A better than best-case estimate is usually an unrealistic estimate that has a causes more strain and problems than the best-case estimate and has many underlying assumptions with it. For example, a PM might set barely reachable standards for a product deadline. However, this might come at the cost of overworking the employees with 60-100 hour work weeks (which is not uncommon at a company like Tesla and SpaceX). These assumptions are really not good because assuming people can work this hard for this long can cause burnout, fatigue, and in SpaceX's case, a bunch of failed rockets. This expectation is just unrealistic and so demanding that it becomes worse than the best-case estimate.

- 8) In your opinion, when would be the best time to tell your customer that you will NOT be able to meet her delivery schedule? Why do you feel that is the best time? Do you think that would be a difficult conversation?
    - The best time to tell the customer that the delivery schedule will not be met **is as soon as possible.** Keeping any secrets or letting it shift is the worst possible thing that can happen, especially if the customer finds out that it was already known. Having transparent communication with the customer is key, and one should not be afraid to let them know if the schedule is behind–people are humans and mistakes happen. The earlier the problem is told, the earlier it can be rectified as well. That way everyone stays on the same page and the problem can be mitigated as possible.
- 9) Discuss why you think branching in your software configuration is bad or good. Describe a scenario to support your opinion.
    - Branching is a key configuration that is extremely good and beneficial for all development teams. Especially with the advent of dev, staging, and prod

    branches, having a branch system means that you are responsible for your own code that you work on, assuming that the master branch you pull from is tested correctly. Having features be independent of each other means that any bugs can be tracked and maintained much more efficiently, as well as people not having to worry about breaking other people's code. It is essential for any large-scale and enterprise software, especially having staging and prod environments. That way, code has to go through rigorous testing from multiple developers before it can get pushed towards customers. When code gets merged in, companies like Github provide very helpful tools for merging the code in successfully. Without branches, codebases would be much harder to maintain and much more prone to having bugs and crashes.

- 10) Have you used a build tool in your development? Which tool have you used? What are its good points? What are it's bad points?
    - Yes, we use a build tool called Github Actions that does continuous integration and testing for our project. Its good points are that it ensures that only successful builds pass into our Github repository and that all tests pass for every branch. It has no real bad points as continuous integration is a standard for all good development practices.