Serena Zafiris
Dr. Johnson
CMSI 401
October 30th, 2019

Assignment 2

Problem 1

Write a short paragraph to answer these three questions:

- **What are the two major concerns of any software project?**
    - The two major concerns of a software project are how long will it take to complete the project and how much will it cost?
- **Which do you feel is more important?**
    - I think how long a project will take is more important because the cost is directly dependent on that. The size of the project and the length to completion directly affects how many people will need to be hired and how long you will have to pay them.
- **Where does the idea of complete functionality fit with these two concerns?**
    - Complete functionality directly affects these two concerns. Think of it as you can have two of the three following: complete functionality, on time, and cheap. If you want it to have complete functionality and to be cheap, odds are it will take longer since you will not be able to hire enough people to do it quickly. If you want complete functionality and it to be on time, odds are it will cost a lot more due to the personnel requirements. Depending on the requirements necessary for complete functionality, the time it takes to complete and the overall cost of the project will vary greatly.

Problem 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.**
    - The four main phases of Agile development are requirements, design, code, and then test. I think requirements should try to be as done as possible before the iterations and only be updated when necessary. This way you have a better idea on how to break up the project into sprints which can save a lot of time and money in that you know exactly what to plan for. This does not mean you should not update requirements as you

go, but the more information you have in the beginning, the better the plan you can make, and then make the necessary adjustments along the way. This way you do not lose time on ignoring things that need to be in the software and having to add them last minute.

Problem 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.**
    - The main phases in Waterfall development are requirements, design, code, test, deployment, and maintenance. Waterfall differs from Agile in that maintenance on the code is done after the full deployment of the code. The Agile development process stops after testing. I think maintenance is needed in Waterfall because everything is done over a long time, so there is a greater risk of issues and bugs to occur. Agile may have to do maintenance if after a sprint there is a bug found in the code or the customer needs to change a bit about the functionality. In this case, the maintenance would become a new ticket in the following sprint.

Problem 4

Write one-sentence answers to the following questions:

- **What is a “user story”?**
    - A user story is typically used in agile development where people draft a description of an interaction with the software from the customer’s point of view and how the customer imagines their interactions with the software you are making for them.
- **What is “blueskying”?**
    - Blueskying is when the customer and the contractor get together in a grand brainstorming session and document every one of the customer's big ideas so everyone can see all of the customer's needs and potential needs.
- **What are four things that user stories SHOULD do?**
    - They should describe one (and only one) thing that the customer requires the software to do
    - They should be written in a way that is understandable to the customer
    - They should be written by the customer
    - They should be short, no more than 3 sentences.

- **What are three things that user stories SHOULD NOT do?**
    - They should not be a long essay.
    - They should not contain technical words and phrases that might be unfamiliar to the customer
    - They should not mention specific technologies

Problem 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 do not necessarily agree with this completely. I think assumptions are inherently risky, however, they may be necessary to make progress. While it is good to check in with a customer and make sure you clearly understand what they want, sometimes the customer is going to be either unsure about what they want, or completely unresponsive. You may have to draw your conclusions and hope that they return a positive result. I think assumptions are only good when in dire situations such as the ones listed above.
- **A big user story estimate is a bad user story estimate.**
    - I agree with this sentiment. If you think too large and cause something to be developed over a longer time, you run the risk of not being able to adapt to changes in plans. The bigger you make something, the less you'll be able to change if a customer decides to change focus. The larger the estimate, the longer it will take to develop, which allows for time for the customer to switch focus or for priorities to change. A user story should stay at 15 days or under so you have the ability to change things up and switch things if problems arise.

Problem 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:** Observation, Role Playing, Blueskying
- **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 blueskying fits into the “I help you get more from the customer” because it gives the customer the space to hash out all of their ideas no matter how seemingly impossible or silly they may seem. It can help you understand the big picture behind what the customer wants and better understand what requirements pop up down the road.

Problem 7

- **Explain what is meant by a better than best-case estimate.**
    - A better than best-case estimate is one that has an unrealistic expectation about what is going to get done and has a lot of assumptions as to how things will go. For example, if a customer wants a programmer to write a website that hooks up to a database, the programmer may say, that it will only take 3 days, but are doing so under the assumption that they can work uninterrupted for at least 12 hours a day for those 3 days and have no issues or problems. There are a lot of assumptions within this estimate, that the programmer will have this much time and that everything will go along smoothly. There are also a lot of assumptions the programmer has not taken into consideration, the customer keeping the same plan, not changing any designs, learning curve for any new software or materials, etc. The estimate is very unrealistic and much more demanding than the best case estimate because it ignores a lot of factors and makes a lot of assumptions where if proven to be incorrect, can greatly change the outcome.

Problem 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?**
    - I think as soon as you know you will be unable to meet the schedule you should notify the customer. This gives you the most amount of time to work with them and reschedule and refocus your plan. It is always better to be prepared and tell people as soon as possible. The more you lead someone on, the worse it will be when you tell them that you cannot meet their demands. I think this will always be a difficult conversation because you will have to disappoint someone. However, honesty is always the best policy, and being clear with someone will always work out much better

than lying to them and getting caught later. You will save the customer a lot of money if they know ahead of time what is going to happen.

Problem 9

- **Discuss why you think branching in your software configuration is bad or good. Describe a scenario to support your opinion.**
    - I think branching, in the end, is a good way to handle multiple people working on the same code at once. While branching has risks of merge conflicts and people using and working on outdated code, all these things can be managed and worked through if proper guidelines are followed. Branching also prevents people from damaging the master code and allows developers to play around with their code without fear of breaking it for anybody else. In the scenario of our senior project Amplify, branching is good because it allows us to work on both the frontend and the backend without stepping over one another. It also again prevents us from breaking the master code. Since the app will eventually be in the app store, we do not want it to be easy for someone to push to master and break the app. Branching will allow us to thoroughly explore and test components before they go to the actual product. This was we can work with our code without breaking the client-facing product.

Problem 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?**
    - For our project, we are using Github Actions which is a tool that allows for continuous integration and testing. It is great in that it ensures the successful guilds are the only ones that pass into the Amplify repository. It also ensures that all tests pass for every branch. I cannot really think about any bad points in the sense that it is not hurting our code, it is rather protecting it. However, it can be difficult in that it requires a lot more work on the development end to make sure your code is up to par. But, I would not count this as a negative because it can save us from future issues that pop up due to improper code.