Alexia Filler
Dr. Johnson
CMSI 401
10/30/2019

Assignment 2

Problem 1

Write a short paragraph to answer these three questions:

- **What are the two major concerns of any software project?**
- **Which do you feel is more important?**
- **Where does the idea of complete functionality fit with these two concerns?**
    - The two major concerns for any software project are *cost* and *duration* (how long it will take to complete). I feel that cost is the more important of the concerns because, in many cases, the budget of a project is what challenges a company most. While it is often acceptable for the release of a project to occur slightly later, it is harder to find leniency for monetary costs. Plus, if the deadline for a project is extremely strict but the amount of work to be completed before the deadline is not manageable, it will require longer hours and perhaps more workers on the project, which, still, will accrue greater costs. Complete functionality entails that a project is completed and fully functioning using less than or equal to the budget than was allotted for it in less or equal to the time than was allotted for it, while still maintaining the quality of the project. This ties in with these concerns because it alleviates stress regarding them; the project is completed without having to go over budget or extending the deadline!

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 that occur in every iteration of the Agile method are requirements, design, code, and test. I think that, for the sake of maintaining the quality of the project, steps should not be omitted in iterations. With every new step, task, or feature, it seems important to fulfill every step, especially since work that has been done thus far might affect work that will be done in the following steps. While I believe that the requirements should probably be outlined at a higher level at the beginning of the entire process, it still seems important that each iteration lay out its specific requirements as well. I think that the amount of time that it would save to skip the requirements step (which seems to be the least vital step) would not be significant enough to risk missing some important, necessary element.

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 be needed.**
    - The main phases of the Waterfall method of software development are requirements analysis, design (product design and system design), coding/development, testing, and maintenance. The phases of the Agile method are requirements, design, coding, and testing. Therefore, Waterfall has the maintenance step, while Agile does not. I think that the maintenance phase is needed in Waterfall because, in this method, the process is passed through one time and the developer does not move on to the next step until the current step is fully completed. Each step is thus only completed once, in its entirety. Maintenance is then required so that adjustments can be made after the deployment of the product. In comparison, in Agile, the cycle is gone through many times so that parts of the process can be updated as needed throughout the cycle. I think that, generally, Agile does not need the "maintenance" step because, by virtue of its cyclical design, updates are made as needed. However, if upon deployment of the product, it is expected that, in the Agile method, more updating is not required, then maintenance could be useful, because, as a program is used over time, unforeseen bugs can reveal themselves as technologies change (platform of use, operating system updates, etc.).

Problem 4

Write one-sentence answers to the following questions:

- **What is a "user story"?**
    - A user story is the description from the customer's point of view of how they imagine interacting with the software you are producing for them.
- **What is "blueskying"?**
    - Blueskying is a big brainstorming session between the customer and contractor where every one of the customer's grandest ideas are documented so that all the potential needs of the customer are laid out in front of everyone.
- **What are four things that user stories SHOULD do?**
    - All from pg 39:
    - They should describe one thing that the customer needs the software to do
    - They should be written in language that the customer understands
    - They should be written by the customer
    - They should be concise
- **What are three things that user stories SHOULD NOT do?**
    - All from pg 39:
    - They should NOT be long
    - They should NOT contain technical jargon that the customer might not understand
    - 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 think it is right to say ALL assumptions are bad. I believe there are some fair, although possibly obvious, assumptions that a developer can make. For instance, the developer can probably assume that the contractor would like the code to run as efficiently as possible in terms of both time and space complexity. Besides this, perhaps making assumptions related to things that the contractor did not necessarily specify is necessary at times, although not preferred. But I think I would say this only in regards to small decisions, such as picking a color palette for an app if it was not specified to begin with… Something that can be changed extremely easily. Beyond that, though, I would definitely say it is far better to compile a list of questions to go over with the contractor as soon as possible instead of assuming anything major about design or functionality.
- A big user story estimate is a bad user story estimate.
  - I agree that a big user story estimate is a bad user story estimate. Having long estimates for when a big piece of functionality will be done makes it more difficult to set checkpoints to work on. According to the book, a user story estimate should fit within one iteration of work, since estimates that are much longer decrease in accuracy. It is better to break a big user story up into smaller, more manageable goals, so that, if the ideas for the goals change, only a few weeks' worth of work will need to be altered, instead of several months'.

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*:
  - 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.

    - My answers mostly agree with the book's answers. However, I think that Blueskying should be included in "I help you get more from the customer" because Blueskying is essentially where you get EVERYTHING from the customer all at once. All of the customer's ideas get laid out in Blueskying, so it seems like you definitely get a lot from the customer in that process.

Problem 7

- **Explain what is meant by a better than best-case estimate.**
    - The best-case estimate for a project is even optimistic, creating a deadline that outlines how the progress will go without any unforeseen hiccups. A better than best-case estimate, therefore, is generally unrealistic, because it takes that best-case estimate and tightens the deadline even further. It implies a great deal of strain on the developer, and still assumes that there will be no unforeseen hiccups. It is much better to create a realistic, thought-through outline of the work that needs to be done so that the work can be done within regular working hours without expecting an unreasonable amount of work from the developer, while also maintaining that the deadline can be upheld.

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 it would be best to let the customer know that you will not be able to finish with the expected tasks within the bounds of the delivery schedule as soon as such an issue becomes apparent. This is because I imagine that transparency and good communication are vital in the customer-contractor relationship. I think it might be a less comfortable conversation, since you are delivering somewhat bad news. But it is better to be honest early on than to get to the very end and fail to deliver on a more important deadline.

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 is good in a software configuration. For example, we have been using branching in our senior project, Amplify. While branching does leave room for issues like merge conflicts or having people continue working on their branches without refreshing to the newest version of the code, it also lets multiple people work on different parts of the code at once. This makes us more productive, because we can complete multiple tasks all at once by splitting the workload. It also keeps us organized, because we make sure to follow good branching etiquette (naming our branches appropriately and descriptively). Plus, and perhaps most importantly, branching lets us work without having to affect the code on our Master branch, which holds the most up-to-date working code. This ensures that new code must go through review to make sure it is working and will not break anything in our Master code before it gets pushed, so our Master branch will only ever hold *working* code.

Problem 10

- Have you used a build tool in your development? Which tool have you used? What are its good points? What are its bad points?
    - In our development, we have been using GitHub Actions as our build tool. Its good points are that it allows for continuous integration of our code, which ensures that broken builds will not pass into our repository. I cannot identify any of its bad points; the continuous integration is helpful and easy to understand because of the friendly and easy-to-use interface. GitHub itself contains so many helpful tools!