Course Professor: Dr. Johnson
Team Members: Breelyn Betts, Maya Dahlke, Adriana Donkers, Henno Kublin, Lexi Weingardt
CMSI 401
28 October 2020

Pilone Book Written Assignment

<u>Problem 1:</u> 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?**

When it comes to software development, the main concerns of any specific project would be both the cost and the time it would take to develop the project. Personally, we think that the time it takes to develop a project is more important, but that could be a bias since all of the projects we have worked on so far in college have not had money as a component. However, every time a big project comes out and fails, especially with video games, the common explanation by the developers is that “there was simply not enough time” and rarely a debate about the budget. The idea of complete functionality fits into these two concerns because complete functionality is what is to be achieved if both of these concerns are held at bay. If both the budget and time are handled correctly and as planned, then the final project should have complete functionality.

<u>Problem 2:</u> 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.

In Agile development, each iteration of a project has a requirements, design, code and test phase. The requirements phase involves planning what the iteration will accomplish and the time frame it will take. The design phase sketches out the architecture for this specific feature. The code phase implements this design and functionality in the actual code. The test phase ensures that the feature functions properly and matches the outlined requirements in the initial phase. Once this iteration is complete, feedback is requested from the client. Because each of the phases in each iteration are dependent on one another, each phase should be completed in every iteration. Additionally, each of these stages is essential to produce clean, working code. Although it may save some time upfront to omit one of the phases, a developer will spend more time fixing bugs or making adjustments at the end of the project if any of these steps are skipped.

<u>Problem 3:</u> 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 that occur in the Waterfall method for software development are: requirements, design, development, testing, and deployment in that order. Unlike the phases in Waterfall, which are linear, the phases in Agile development occur in multiple cycles called sprints. With agile, each time you complete a cycle, you get a cumulative outcome (feedback) and work off of that. Waterfall doesn't require cumulative outcomes because there is more upfront planning and detailed documentation that happens before executing the main phases. So, instead of multiple cumulative outcomes, it only has one big outcome at the end. Since Agile is more flexible, it requires a lot of collaboration, communication, and trust between team members. If we are working on a project using the Agile methodology and the team is not communicating effectively, then the team might decide to use the Waterfall phases of drawing up documentation on who gets what done and when. This would allow for a more predictable timeline.

<u>Problem 4:</u> Write one-sentence answers to the following questions:

- **What is a "user story"?**

A user story is a portion of what you will build for your project written from the customer's perspective about how they will interact with the software built.

- **What is "blueskying"?**

Blueskying is when you brainstorm big with others about the project ideas or requirements.

- **What are four things that user stories SHOULD do?**

User stories should describe one thing that the software needs to do for the customer, be written using language that the customer understands, be written by the customer, and be short.

- **What are three things that user stories SHOULD NOT do?I**

User stories should not be a long essay, use technical terms that are unfamiliar to the customer, or mention specific technologies.

<u>Problem 5:</u> 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.***
- ***A "big" user story estimate is a "bad" user story estimate.***

We believe that, in terms of formulating an estimate for a user story and ultimately an entire project, assumptions are never good. You need to weed them out in order to minimize possible setbacks and issues in completing your project. However, we acknowledge that sometimes it may not be possible to get rid of all assumptions, in fact that is probably near impossible. But the more assumptions you can get rid of, the more closely your project will probably follow your estimates. We don't necessarily believe that a "big" user story is in itself "bad". However, large user stories make it harder to complete the project in a timely manner. It is probably best to stick to the 15 day estimate, but in the case that a user story is very important or significant to the project we believe that more time can be allowed. Past about a month, the timeline for the user story gets a bit unrealistic and difficult to effectively complete.

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:** Role playing
- **The more of me there are, the clearer things become:** Observation
- **I help you capture EVERYTHING:** Blueskying
- **I help you get more from the customer: Estimate,** Observation
- **In court, I'd be admissible as firsthand evidence:** Observation
- **Some people say I'm arrogant, but really I'm just about confidence:** Blueskying
- **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.

We disagreed on two of the answers with the textbook, but only slightly. The first blank we said was “role playing,” as it seems that a role play using the software could be reconstructed into a formal use case, but so could a user story as the textbook states. Secondly, we said that “blueskying” is really just about confidence, which definitely makes sense to me considering it requires developers to be as creative and open minded as possible, and thus confident in their work. The textbook’s answer of “estimate” works as well, but we thought blueskying would be a better fit.

Problem 7: Explain what is meant by a “better than best-case” estimate.

A better than best-case estimate is an unrealistic estimate that a project or feature can be completed in a short amount of time based on some assumptions. A better than best-case estimate over promises a result. The unrealistic estimate will be detrimental for everyone involved with the project. The developer will overwork themselves to attempt to meet a deadline that is unreachable and the project manager and client will be disappointed with the delay. This stresses the importance of realistic expectations of productivity and external factors when planning estimates.

Problem 8: In your opinion, when would be the best time to tell your customer that you will NOT be able to meet their delivery schedule? Why do you feel that is the best time? Do you think that would be a difficult conversation?

If you are 100% sure that you will not be able to meet the customer’s delivery schedule, it is best to tell them as soon as possible. It may be a difficult conversation, since the customer will be disappointed, but a least it will give you and the customer time to come up with a contingency plan to extend the schedule and deliver at the earliest time possible. If you wait until the delivery date to tell the customer, the conversation will only become more difficult because you waited so long.

<u>Problem 9:</u> Discuss why you think branching in your software configuration is bad or good. Describe a scenario to support your opinion.

We think branching is good in some scenarios but can cause damage in other instances. When used correctly, branching can be used for good software development. For example, if you want to release a version of the software that needs to be maintained outside of the primary development cycle, it's a good idea to use a branch to maintain the code. However, if you can accomplish a goal by splitting the code into different files, then you probably should avoid branching.

<u>Problem 10:</u> 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?

The "build tool"' we are using for our project are the command line tools that allow the user to build their project in Xcode. In Xcode, these command line programs are called using scripts, allowing users to make workflows without Xcode being open at all. The command line build tools are useful because users can work strictly from the command line without opening Xcode, as was mentioned above. However, the build can take a very long time and it is not always the most efficient.