Group 01: Charitable
Veda Ashok, Sam Gibson, Rachel Nguyen, Bridget O'Connor, Krysten Tachiyama
Dr. Johnson
24 October 2020

Problem 1

The two major concerns of any software project are how much the project will cost, and how much time it will take to complete. We consider time to be the more important of the two. While cost is important, Pilone and Miles mention that often the customer will want the software for a specific event/date. Therefore the cost may be negotiable and certain features of the project can be cut or added to affect the cost, however the deadline is set in stone.

Problem 2

The four phases of Agile method software development are defining requirements, design, code, and test. We feel that these requirements are required in each iteration for a successful project. In our experience, requirements have shifted and developed as we work on them. Having the option to revisit the requirements and modify them as needed is an important part of developing software. When you change a requirement, there is inevitably a change in design, code, and testing. Hence, it is important to have each iteration be a mini-project. It is why Agile development is so effective!

Problem 3

The main phases of the Waterfall model in software development are requirements analysis, design, code, test, and maintenance. The difference between Agile and Waterfall is that the Waterfall model is a linear process while Agile is iterative, or cyclical. In Waterfall, each of the phases represent a distinct stage of development where each stage finishes before the next begins. Agile does not have a maintenance phase. Since Agile is iterative, the maintenance happens as we develop the software. With a Waterfall model, all of these phases are important since it is a matter of

developing the requirements linearly. Waterfall does not have as much opportunity to test and modify the design and requirements throughout the project. It requires a more concrete understanding of what we want the software to look like from the start. Of course, that is not to say that we can never modify our design, but it takes more work because it requires "going backwards." In Agile development, maintenance may be required for a project where the deadline was before the project was completed. In this case, developers may choose to release an earlier version of the product and then continue to develop new features that would be released as updates.

Problem 4

Write one-sentence answers to the following questions:

What is a user story?

A user story is a description of a feature of the software that is written from the perspective of the user.

What is blueskying?

Blueskying is the process of brainstorming with others, allowing ideas to flow without any judgments.

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 the language the customer understands,
be written by the customer, and be short.

What are three things that user stories SHOULD NOT do?

A user story should not be a long essay, use technical terms that the customer may be unfamiliar with, or mention specific technologies.

Problem 5:

1. In the context of software development, we agree that "all assumptions are bad, and no assumption is a good assumption." Assumptions leave room for error and misinterpretation so it is better to ask what the customer wants, then do what they say, and if you are unsure then ask further questions. Unfortunately

assumptions may be a necessary evil if the customer can not be reached in time. You know what they say when you assume...

2. We also agree that "a big user story estimate is a bad user story estimate" because a large user story can be further broken down into smaller ones. A user story that is too big would either take extra iterations or require that one iteration be extended in length. This would defeat the purpose of agile development in creating small, achievable tasks that can be done in a short period of time. Having a big user story may require longer sprints or sprint extensions, and we do not want to delay development when we have promised our customer a deadline!

Problem 6:

a. You can dress me up as a use case for a formal occasion: User story
b. The more of me there are, the clearer things become: User story
c. I help you capture EVERYTHING: Blueskying, Observation
d. I help you get more from the customer: Observation, Role Playing
e. In court, I'd be admissible as firsthand evidence: Observation
f. Some people say I'm arrogant, but really I'm just about confidence: Estimate
g. Everyone's involved when it comes to me: Blueskying

Problem 7:

A "better than best-case" estimate is an estimate that assumes that nothing will go wrong in development. It may also assume that less work needs to be done than necessary (e.g. assume no need for testing). This means that the "better than best-case" estimate is an unrealistic estimate.

Problem 8:

In our opinion, you should tell the customer that you cannot meet their delivery schedule as soon as you have an idea of what you can achieve by the deadline. When you tell the customer that you cannot meet their schedule, you should at least be able to tell them what you can accomplish in that time; you should have a plan. This will be

a difficult conversation to have- the customer will likely be unhappy to hear that they will not receive what was originally agreed on. However, giving them an idea of what they can expect is professional and shows the customer that the effort is being made to be transparent and productive.

Problem 9:

We think branches are good for development because they allow us to add features and make changes to our code in a way that does not affect the other developers. For example, if we want to work on a new feature, we can create a branch to do so. While we are working on our own branches, each person can feel comfortable even if one of us temporarily breaks the code since it will not affect the other developers. Then when we finish working on the new feature, we can merge it back into master so that the other developers can see a new working feature in the master branch. This way we can ensure that the master branch is always functional while we work on features because only working features will be added to master. Additionally, it helps when working with a team because we do not have to keep checking with others to pull or commit changes.

Problem 10:

A build tool we use in the development of Charitable is Vercel, and it is pretty neat. Vercel deploys branches when they are pushed into the preview environment so that we can easily see if we are committing breaking changes. Each commit on a non-master branch gets its own URL. We have integrated Vercel with GitHub so that when we push to the repo, it will show a red X or a green check depending on whether or not the change breaks the build. There are also different environments which can hold different environment variables. By default the master branch is the production environment so each push to the master branch does not get its own url but rather is deployed to the current site domain. This allows us to have continuous deployment.

This is a major pro in the way that the moment we push a change to our site it is deployed. This can also be a con in the way that maybe a change we make is not a breaking change but it is not quite one that we want, but we end up pushing it. In this way we might end up effectively breaking the site the moment we push. For example, we pushed a change that broke our API call but Vercel deployed it anyway since it doesn't check if api calls work before deploying since it just checks if the site builds properly. Though, of course, we can also go back to a commit to remedy the change. Also if it is a very obvious breaking change, like the site will not build, then Vercel will not deploy that update and it will notify you that you are trying to upload a breaking change.

Another con is that for safety it hides the env variables the moment you input them, so if you think you put in the env variable incorrectly, you need to delete it and then re-add it. It is also very difficult to have a continuous deploy if your app is not at the root level. Lastly, you have to pay if you want a team to all have access to a domain, but you can get around this if your team shares one account.