Pilone & Miles Reading Assignment
Owen Dewing, Kelly Tao, Juan Ballesteros, Joanna Estrada

1. The two major concerns of any software project are how much it will cost, and how long it will take. I feel like time is more important than cost. You could have a brilliant idea, but if you take too long to flesh out that idea into a piece of software, somebody could get to it first, or that issue might not be relevant anymore. Complete functionality interacts both with cost and time, and this can be modeled as a triangle connecting cost, time, and functionality. For example, if a project is created quickly and with low cost, there will be less functionality. If a project is created quickly and with full functionality, it would typically be expensive. Finally, if a project is created with low cost and full functionality, it would take longer to make as fewer resources are utilized to complete the project.

2. The five main phases that occur in Agile Development are brainstorming/planning, designing, developing/coding, quality assurance/testing, and maintenance. I think that if the team knew *exactly* what they wanted their project to look like from the very beginning, then they could try to accomplish the brainstorming/planning phase at the start of the project, but I don't think that's very realistic because certain aspects of your project always change as you develop it, and you can't predict everything from the beginning.

3.
   The main phases in the waterfall method are requirements analysis, design, code, test, and maintenance. These are very similar to the phases in Agile development, except for the additional reflection phase Agile method has to measure the progress of the team and to make adjustments based on that. One phase that happens in waterfall method but not Agile method could be Critical Design Review, since in Agile method, there is a rather frequent meetup with customers, leaving more time and space for them to make changes to designs, than a final check for all designs. But in waterfall, since coding usually happens when all design is almost finished, it is necessary for a final check on the design so everyone can work on it fully. A situation I could think of Agile method having this review is when the application desired by the user is very specific and with minimum blur areas, and with little options the design team could come up with some final design in early cycles.
4.

- What is a "user story"?
  - A user story is a story written in the customer's language about how their users interact with the software you're building.
- What is "blueskying"?
  - Blueskying is the process of thinking big and not ruling out any ideas in the requirements phase.
- 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 a language that the customer understands, be written by the customer, and be short.
- What are three things that user stories SHOULD NOT do?
  - User stories should not be a long essay, use technical terms that are unfamiliar to the customer, and mention specific technologies.
- Does the Waterfall method have "user stories"?

- No, the waterfall method does not use user stories. User stories are used more in Agile software development.

5. “All assumptions are bad, and no assumption is a ‘good’ assumption”

I disagree with this statement. Sometimes assumptions are necessary to move your project forward because you don’t typically have all of the complete information available. Assumptions can be good and streamline decision-making.

“A big user story estimate is a bad user story estimate”
I mostly agree with this statement, because big user stories could be too complex for the scope of your project. Smaller user stories are more specific and concise, and allow the team to deliver a product with more accuracy.

6.

- 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: Observation, Blueskying
- I help you get more from the customer: Observation, role playing
- 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

7. A “better than best-case” estimate is an overly optimistic estimate on how long a certain task will take. The developer assumes that nothing will go wrong, and that they are the only developer working on that specific task, leading to an unrealistic and rarely achievable estimate.

8. If I had a customer and I realized that I wouldn’t be able to meet their delivery schedule, I would try to tell them as soon as I realized that delivery wouldn’t be possible. This early communication would allow the customer to adjust their plans, and would maintain some of the trust between the customer and I. If I waited to tell them until the last minute, this would invoke panic and frustration. I think a way to make the conversation less difficult would be to come prepared with new ideas and alternative solutions to offer to the customer.

9. I think that branching in software development is a good idea, but if left unchecked, could lead to bad results. Branching is beneficial because it allows for parallel programming, and multiple people can work on different features without affecting the main codebase. Also, if you encounter a bug in a branch, it won’t affect the main codebase. Branching can lead to bad things if a branch is not merged to the main codebase for a while. This could lead to a lot of merge conflicts with the main codebase and could confuse other developers if you are developing with a team. A potential scenario could be working on a group scheduling app, and one person could have a

branch working on integrating the API and another person could be working on a branch implementing firebase/firestore. This would allow for both people to program independently and would lead to more progress with the app. However, if the person working on the API branch never merges it to the main branch, their changes could become outdated as the main branch changes, leading to potential merge conflicts and unnecessary code. It is important to merge your branches often to update partners on your progress and to keep the main branch up-to-date.

10. I have used CMake before. The good point of using it is that it is a powerful tool that could run big projects that have many dependencies in it in any environment, with a rather clear oversight. But I really dislike it, especially with its documents given, which is basically none. The CMakeLists is very hard to understand for beginners, the namings such as BINARY_DIR, CMAKE_BINARY_DIR and PROJECT_BINARY_DIR are just nonsense at first glance, and their document literally says nothing. Also, it is set in a way that only one CMakeLists could be run for a single folder. I could not run part of CMake, do some command line operations, then come back to CMake, instead I have to learn all the new commands in CMake to do those command line actions, which are written in very confusing grammar and language. Also, on many occasions, CMake is weak on finding packages. For example, I was using QT the other day, and though I have QT installed my CMake couldn’t find it, I have to do many extra steps such as going into global config to manually add the path of QT’s “bin\” into user’s environmental settings.