CMSI 4071: Pilone & Miles Book Assignment
Caroline Ellis, Jack Seymour, Breea Toomey

1. Write a short paragraph to answer these three questions:
    a. What are the two major concerns of any software project?
    b. Which of those two do you feel is more important?
    c. Where does the idea of complete functionality fit with these two concerns?

    The two major concerns of any software project are how much it will cost and how long it will take. Of these, I believe that cost is the more important concern. While time is definitely a factor to be mindful of. As most clients have deadlines and events that they want the project ready for. Underestimating costs can have much more serious consequences. If the project runs over budget, it can completely drain all resources and lead to a loss and/or inability to complete the project at all. The idea of complete functionality fits into these concerns because achieving all of the functionality desired characteristics takes both time and money. Adding more features will increase expenses and extend development time.

2. Write a short paragraph to answer these three questions and briefly explain your opinion:
    a. In the Agile method for software development, what are the five main phases that occur in each and every iteration?
    b. Do you feel that any of them could be done at the start of the project and not be repeated in every iteration?
    c. Do you feel that would save time overall on the project?

    In the Agile method for software development, the five main phases that occur in every iteration are: Brainstorm, Design, Development, Quality Assurance, and Deployment. While it might seem that Brainstorm and Design could be done at the start of the project and not repeated in each iteration, this approach assumes that all potential roadblocks are anticipated and the initial design is flawless. This seems to be an unrealistic expectation in most cases. Although skipping these phases in future iterations could theoretically save time, it would risk missing critical adjustments and improvements. Agile's iterative approach can actually save time by allowing the team to continuously refine and adapt to the needs of the project.

3. Write a short paragraph to answer these four questions and briefly explain your opinion:
    a. In the Waterfall method for software development, what are the main phases that occur?
    b. How are they different from the phases in the Agile method?
    c. What other phases are in Waterfall that are left out of Agile?
    d. Do you think these are needed in Waterfall?
    e. Describe a situation using Agile in which one of these extra Waterfall phases might be needed.

The Waterfall method can be described as a linear software development methodology in which each phase must be completed before the next begins. These phases include: requirements gathering, design, coding, testing, and deployment. On the other hand, in Agile methodology, development occurs in iterations, with each iteration producing a working product. Agile prioritizes flexibility, collaboration, and continuous improvements over these rigid phases (typically 2-3 week sprints). Waterfall's extra phases that are usually left out of Agile include detailed documentation and planning prior to development. While these can be valuable, Agile often finds them excessive. I think that in a complex project with stringent regulatory requirements, Waterfall's detailed documentation may be essential to ensure compliance considering the lack of development, test, and review iterations. For example, an Agile development situation which might require these extra documentation and planning steps may be a large-scale healthcare software project that must adhere to strict compliance regulations. In a case like this, comprehensive documentation and planning is crucial.

4. Write one-sentence answers to the following questions:
    a. What is a user story?
        i. A user story is a requirement written from the customer's perspective in their language, capturing a single interaction between the user and the software you're building.
    b. What is blueskying?
        i. Blueskying is a collaborative brainstorming technique where you iterate with the customer to explore their requirements and think big, encouraging everyone to share all ideas without judgment in order to capture the full breadth of possibilities.
    c. What are four things that user stories SHOULD do?
        i. Describe one thing that the software needs to do for the customer.
        ii. Be written using language that the customer understands.
        iii. Be written by the customer.
        iv. Be short (Aim for no more than three sentences).
    d. What are three things that user stories SHOULD NOT do?
        i. Be a long essay.
        ii. Use technical terms that are unfamiliar to the customer.
        iii. Mention specific technologies.
    e. Does the Waterfall method have user stories?
        i. No, the Waterfall method's rigid, linear structure doesn't allow for the iterative and flexible approach required for using user stories.

5. What is your opinion on the following statements, and why do you feel that way:
    a. "All assumptions are bad, and no assumption is a good assumption."
    b. "A big user story estimate is a bad user story estimate."

I think that both of these statements offer valuable insights into effective project management. From my analysis, the first quote emphasizes the importance of critical thinking and avoiding biases. By recognizing that all assumptions carry a degree of uncertainty, software development teams can approach problem-solving with a more open and objective mindset, helping to reduce the risk of making costly and/or dangerous mistakes based on faulty assumptions.

The second statement seems to highlight the need for more granular user stories when working in Agile development. From my knowledge/experience, large ambiguous user stories can lead to scope creep, missed deadlines, and overall decreased team morale. By breaking down work into smaller, more manageable tasks, teams can better estimate effort and track all progress made.

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.
    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: _____Role playing, Observation_____
    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_____

7. Explain what is meant by a better than best-case estimate.
    a. A better than best-case estimate refers to an overly optimistic prediction that a programmer gives for how long it will take to complete a task. It usually is under the assumption that everything will go as smoothly as it theoretically can. This can overlook possible realities like: the need for testing, communication and coordination with other team members, human error, and human needs for rest and break time. This estimate is usually incorrect and the actual time needed is longer than anticipated.

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? If so, how could you make it less difficult?

In my opinion, the best time to tell a customer of an anticipated missed/extended delivery schedule is as soon as it is known that a deadline will be missed. From my knowledge and experience, transparency and honesty build trust and allow the customer to adjust plans on their end as needed. Delaying any notification leads to frustration and potentially damages the business relationship. This would be a difficult conversation, however, proactive communication would help to mitigate any negative impacts. Along

with this, I believe it's essential to apologize sincerely, explain the reasons for the delay, and offer solutions or potential alternatives. Providing a revised timeline and regular updates would further help to make the conversation less difficult by maintaining the customer's confidence and minimizing disruptions.

9. Write a short paragraph to discuss why you think branching in your software configuration is bad or good, then describe a scenario to support your opinion.
    a. Branching in software configuration is a powerful but double-edged tool that requires thoughtful application. On one hand, branches provide a safe space to experiment with code changes, maintain different software versions, and allow teams to work independently without affecting the main codebase (trunk). The ability to isolate changes and test risky modifications without impacting production code is vitally important. However, branching comes with significant responsibilities, as each branch requires maintenance, testing, and careful synchronization with the trunk, which can quickly become overwhelming if not managed properly. Despite these challenges, I believe branching is ultimately beneficial when used carefully and for the right reasons. For instance, when maintaining a legacy version of software for specific customers while developing new features in the trunk, branching is essential. The key is to branch only when absolutely necessary, such as for supporting released versions or testing major experimental changes, rather than as a workaround for poor development practices or team coordination issues. By treating branching as a strategic decision rather than a routine convenience, teams can harness its benefits while minimizing the maintenance overhead that comes with managing multiple code versions.

10. Have you used a build tool in your development? If you have, which tool have you used? What are its good points and bad points — in other words, what do you like about it and/or dislike about it?
    a. NPM as a build tool has both strengths and limitations in my experience. On the positive side, it has a massive collection of packages, straightforward dependency management through package.json, and excellent documentation. I particularly appreciate how NPM scripts can be chained together to create complex build processes using simple commands. But there are also drawbacks, such as how NPM can be slow when installing large numbers of dependencies, and the node_modules folder can become extremely large, consuming significant disk space. Additionally, managing conflicting dependencies can sometimes be challenging, especially in larger projects with many dependencies. Despite these limitations, I find NPM's development server with hot reloading to be a very helpful tool that has notably improved my development workflow and productivity. For example, the 'npm run dev' command enables real-time feedback, letting me instantly visualize how code changes affect the website's appearance and functionality without manually refreshing the browser. This hot-reloading

capability has significantly transformed my development process, making it more efficient and interactive.