https://bjohnson.lmu.build/cmsi4071web/assignmt2.html

Problem 1

Write a short paragraph to answer these three questions:

- What are the two major concerns of any software project?
- Which of those two do you feel is more important?
- 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. Between the two, cost is more important because without sufficient funding, the project cannot move forward regardless of the timeline, and running out of resources can delay or even stop development entirely. Complete functionality means that the software performs all the tasks it was designed to do effectively. Achieving this requires both time and money, but it's crucial to balance these factors by setting realistic goals and expectations so that the project delivers its essential features within feasible limits.

Problem 2

Write a short paragraph to answer these three questions and briefly explain your opinion:

- In the Agile method for software development, what are the five 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?

In the Agile method, the five main phases that occur in every iteration are Brainstorm, Design, Development, Quality Assurance, and Deployment. Repeating each step in each iteration is essential to adapt to changing requirements and user feedback and should not be skipped. Skipping these steps early on may appear to save time, but it often leads to more problems later, such as unclear goals or poor implementation. Although seemingly skippable at first glance, skipping a step like Brainstorming will

limit creativity and prevent the team from identifying new or improved solutions based on feedback from previous iterations.

Problem 3

Write a short paragraph to answer these four questions and briefly explain your opinion:

- 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 in the Waterfall method are requirement analysis, product design, system design, coding, testing, and maintenance. They're different from Agile because Waterfall is more step-by-step and not as flexible, while Agile is iterative and allows changes throughout the process. The phases left out of Agile are requirement analysis and system design. These are important in Waterfall because requirement analysis helps the team understand exactly what to build, and system design shows how all the parts will fit together to meet those requirements. A situation where system design might be needed in Agile is when creating software for high-risk systems like medical devices or air traffic control, where safety and accuracy are critical and everything has to be clearly planned out beforehand.

Problem 4

Write one-sentence answers to the following questions:

- What is a user story?
    - **A user story is a brief, simple description of a software feature from the perspective of an end user, focusing on user needs and goals in agile development.**

- What is blueskying?
  - **Blueskying is the practice of freely brainstorming innovative ideas or solutions without considering feasibility or constraints.**
- What are four things that user stories SHOULD do?
  - **User stories should clearly articulate user value, be concise and understandable, allow for collaborative refinement, and be testable with acceptance criteria.**
- What are three things that user stories SHOULD NOT do?
  - **User stories should not prescribe detailed technical implementations, specify rigid solutions, or attempt to cover all requirements at once.**
- Does the Waterfall method have user stories?
  - **The Waterfall method does not use user stories, as it relies on detailed upfront requirements and documentation rather than iterative, user-centered development.**

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.

A big user story estimate is a bad user story estimate.

Our opinion on these two statements is that they are extremely general, and don’t account for cases where there may be a good assumption or a big user story is necessary for the scope of the project. However, using them as a rule that only should be broken in certain scenarios where you have enough information to justify breaking them is probably for the best

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**
- I help you get more from the customer: **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**

We mostly agree with the book answers, we think that role playing doesn't give a truly authentic experience of how an end user would interact with a product, and therefore it should not be included in helping us get more from the customer. This is because stakeholders and developers will have bias toward how we intend for the product to be used rather than having a clean slate and interacting with the software as-is.

Problem 7

Explain what is meant by a better than best-case estimate.

A "better than best-case estimate" is a projection of how long something will take that assumes that everything will go so smoothly, that even without any hiccups and with everything going completely to schedule, there will be unexpected positive developments and efficiencies that take the estimate beyond what any informed person thinks is possible.

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

In our opinion, the best time to tell the customer that the delivery schedule will not be met is as soon as possible. We believe it is the best time because it gives the customer the ability to either find someone else to fulfill the job, or change their expectations with as much good grace as possible. It would certainly be a difficult conversation, but having it in a way where you articulate the reasons as to why and offering a follow-up plan would make it less difficult and allow the customer to be more receptive.

Problem 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.

Branching in the context of our smaller development team with a trivial mobile app implementation is a good idea, because merge conflicts and possible hiccups can be very easily resolved because of the small number of branches. Also, in a smaller development team, communication is more streamlined and it is easier to plan ahead for how changes will have a ripple effect.

Problem 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?

No, we have not used build tools in our development. Build tools offer many advantages in software development, primarily by automating repetitive and error-prone tasks such as compiling source code, running tests, managing dependencies, packaging applications, and deploying software. This automation greatly enhances developer productivity, as it saves time and effort, ensuring that builds are consistent and reproducible across different environments and team members. However, the complexity of build scripts may increase over time, making maintenance difficult. In certain cases, if not properly managed, build tools can add overhead or integrate poorly with other systems, potentially complicating workflows.