* 1. **What are the two major concerns of any software project?** The 2 major concerns of all software projects are the cost of the project and the time it will take to complete the project. The cost of the project needs to be kept in constant control as it could easily explode beyond projection. Time is also something that is a major concern because the complexity of a software project tends to lead to extensions of deadlines. A customer could ask for changes to the product, increasing the time and cost. There are a lot of unexpected events that could happen during the development of a software project that would cause a drastic change in the project’s cost and timeline.
  2. **Which of those two do you feel is more important?** For our current project, I feel that time management is the most important factor to our success. Since we do not have a strict budget to manage, our main focus should be on ensuring that the project is completed on schedule. Managing our time effectively and meeting deadlines also helps maintains momentum, allowing time for testing and evaluation after completion. Furthermore, since this project is done alongside other projects and classes, it's even more critical to have a proper schedule and finish on time.
  3. **Where does the idea of complete functionality fit with these two concerns?** Complete functionality directly influences these two concerns. A product only achieves complete functionality if it meets the customer’s request and needs. Increasing functionality to meet customer’s demand raises project complexity, which extends development timelines and increases costs. On the other hand, a project with a tight budget and deadline would need to reduce features. Complete functionality must be carefully balanced with time and cost to ensure that the final product meets its intended objectives without going past its schedule and budget
  4. **In the Agile method for software development, what are the five main phases that occur in each and every iteration?** The five main phases of every iteration are : Analysing the requirements. Designing prototypes and documents. Developing a product and demo while getting feedback. Identifying defects and resolving bugs and finally the last phase is maintenance.
  5. **Do you feel that any of them could be done at the start of the project and not be repeated in every iteration?** The initial requirements analysis and product design could be done at the start of the project. Since the agile method emphasizes adaptability, these tasks could be revisited and refined as the project goes on.
  6. **Do you feel that would save time overall on the project?** Reducing phases in each iteration will definitely save time on the project to an extent. Although these phases are done at the beginning of the project, it still needs to be revisited and refined at each iteration with new user feedback and features.
  7. **In the Waterfall method for software development, what are the main phases that occur?** The main phases are requirement analysis, product design, system design, coding, testing, and finally maintenance. The requirement analysis phase is done to analyse the scope of the project, understanding customer’s request, and creating detailed specifications. The design phase is done to create the architecture and software design such as component specification. The coding is the creation of the product. The testing phase identifies and fixes defects. The maintenance phase happens after release, performing bug fixes, updates, performance enhancements, and other changes to keep the product functioning.
  8. **How are they different from the phases in the Agile method?** The waterfall is more rigid, with each phase requiring full completion before moving on to the next phase. Agile is more flexible, following an iterative cycle where all phases occur repeatedly in each iteration. Waterfall requires complete and fixed requirements upfront, making changes costly and difficult once a phase is completed.
  9. **What other phases are in Waterfall that are left out of Agile?** Because of Agile's more flexible and iterative method, there are a few phases that are left out from the Waterfall method. The upfront requirement gathering and analysis is not as exhaustive in agile as requirements are gathered continuously and evolve throughout the project. The design phase is also less extensive as designs are developed as new features are built.
  10. **Do you think these are needed in Waterfall?** These phases are definitely needed in waterfall because of the rigid structure. Waterfall’s success relies heavily on completing each phase thoroughly and with clear documentation before moving on. Explicit and detailed documentation in the design and requirement analysis phase ensures that all components work together effectively.
  11. **Describe a situation using Agile in which one of these extra Waterfall phases might be needed.** An agile project that requires regulatory compliance or integration with legacy systems would need these extra waterfall phases. For example, FDA requires extensive documentation and formal requirements specifications. Agile teams often incorporate upfront requirements documentation and design phases to comply with these regulations, ensuring auditability and safety standards.
  12. **What is a user story?** User story is a simple description of the user’s intended interaction and usage of the software.
  13. **What is blueskying?** Blueskying is a creative brainstorming activity aimed at generating innovative, out-of-the-box ideas without any constraints or limitations.
  14. **What are four things that user stories SHOULD do?**
      1. Describe one thing the software needs to do for the consumer.
      2. Written in a language that the consumer can understand.
      3. Be written by the consumer.
      4. Be short, no more than three sentences.
  15. **What are three things that user stories SHOULD NOT do?**
      1. Should not be a long essay.
      2. Should not use unfamiliar technical terms.
      3. Mention specific technologies.
  16. **Does the Waterfall method have user stories?** The waterfall method relies on more comprehensive and detailed upfront documentation that could be made of multiple user stories.
  17. **All assumptions are bad, and no assumption is a good assumption.** I don’t completely agree. Having less assumptions is obviously better because it reduces the risk of being wrong. But sometimes, assumption is also good because it could reduce time and resources. Assumptions are also sometimes necessary when the customer isn’t certain with what they want and need.
  18. **A big user story estimate is a bad user story estimate.** I agree with this statement because the purpose of a user story is to be a simple and concise description of a product’s features. Having a big user story is counterintuitive because it doesn’t achieve its goal. Instead big user stories should be divided into smaller and more manageable user stories that gives a more precise estimate of the product.
  19. **You can dress me up as a use case for a formal occasion:** User stories
  20. **The more of me there are, the clearer things become:** User stories
  21. **I help you capture EVERYTHING:** Observation, blueskying
  22. **I help you get more from the customer:** planning poker, Role playing
  23. **In court, I'd be admissible as firsthand evidence:** Observation
  24. **Some people say I'm arrogant, but really I'm just about confidence:** Estimate
  25. **Everyone's involved when it comes to me:** Blueskying

1. **Explain what is meant by a better than best-case estimate**. A better than best-case estimate is an estimate that assumes everything goes perfect. It is an overly optimistic and unrealistic approach to time and cost prediction that goes beyond the most favorable realistic scenario.
2. **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?** The best time to tell your customer that you will not be able to meet the delivery schedule is as soon as you realise there may be a delay. This allows the customer to adjust their expectations, explore alternatives, and maintain trust. Waiting too long only makes the problem harder to manage. It may be a difficult conversation as it involves admitting that expectations will not be met, which can lead to frustration or disappointment. Clearly explaining the issue and focusing on the solutions will help to make the conversation less difficult.
3. **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.** I believe that branching is good as it allows for better version control management. Branching allows for users to work on new features, and bug fixes without interfering with the main codebase. In this project for example, we are working on two different features: the insider trading algorithm and sentiment analysis. Branching these developments allows for independent testing, reducing risk in the main codebase

.

1. **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?**

Yes, we used a tool for task automation during our development, specifically PM2 configured with cron scheduling to run our daily data pipeline scripts automatically.

What we liked about it: PM2 proved very useful for ensuring our data collection and processing scripts ran reliably every day at the scheduled time without manual intervention. The logging features were also helpful for monitoring the pipeline's execution and identifying any issues. It effectively automated the running of our Python scripts.

What we disliked or its limitations: While PM2 with cron automated the execution schedule, it's not a traditional build tool in the sense of compiling code, managing complex dependencies for distribution, or packaging the final application like Make, Maven, or Gradle would. Its role in our project was strictly as a scheduler and process manager, automating the workflow rather than the build process itself. The success of the automation relied entirely on the individual scripts being correct and functioning as expected.