1. According to Head First Software Development, every software project has two main concerns: how much it will cost and how long it will take. Both are essential, and which is more important often depends on the context. For example, if there is a tight margin for the budget, it is much more important to focus on cost rather than the timeline. However, if you're in a competitive market it's more important to focus on finishing on time so you can stay ahead of the curve. The idea of complete functionality connects these two concerns because both time and cost will end up affecting functionality– it will often be sacrificed if the project is over-budget or behind schedule. Projects have to balance all three things, as all are necessary for a customer to be satisfied.

2. In the Agile method, each iteration includes five main phases: gathering requirements, designing the solution, implementing the code, testing the product, and delivering or reviewing the results with the customer. Some initial planning or setup can occur at the start of the project, such as defining overall goals or architecture, but these phases are repeated in every iteration to allow adaptation to changes and feedback. Omitting or limiting any phase might seem to save time, yet it often results in greater issues later. Repeating all five phases ensures steady progress, continuous improvement, and consistent alignment with customer expectations.

3. In the Waterfall method, the main phases are requirements gathering, design, implementation, testing, deployment, and maintenance. Unlike Agile, Waterfall follows a strict, step-by-step process without returning to earlier stages. It also includes formal documentation and maintenance phases that are minimized or treated more flexibly in Agile development. These additional phases are important in Waterfall because they provide structure, traceability, and accountability for larger or high-risk projects. Agile projects may also benefit from similar phases in regulated industries such as healthcare or finance, where documentation and long-term support are crucial.

4. A. A user story is a short, clear description of a feature told from the end user's perspective, expressing what they want and why.
   B. Blueskying is a brainstorming process used to explore ideas and possibilities without concern for current limitations.
   C. User stories should focus on customer needs, describe what the system must accomplish, remain concise, and support estimation and discussion.
   D. User stories should not describe technical details, dictate specific implementation methods, or be too large or vague to estimate accurately.
   E. The Waterfall method does not use user stories; it relies on formal requirements documentation instead.

5. "All assumptions are bad, and no assumption is a good assumption":
   Not all assumptions are harmful; some are necessary when complete information is unavailable. Assumptions should be identified, documented, and verified early in the process to prevent issues later. The real problem arises when assumptions remain untested or unnoticed.

   "A big user story estimate is a bad user story estimate":
   A large user story often indicates that it is too broad or unclear to estimate accurately. Dividing it into smaller, well-defined tasks leads to better understanding, more accurate estimation, and easier progress tracking throughout the project.

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**

   - Blueskying and Observation might work for this as well since getting more ideas focused on the core needs of a project can be beneficial, and seeing how software would fit into what is being built would make what needs to be done more clear.

   c) I help capture EVERYTHING: **Blueskying and Observation**

   - Role-playing might work as well since it would be good to see what software needs to do (from the customer's perspective)

   d) I help you get more from the customer: **Role-playing and 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**

   - Role-playing might also work since the customer and the developer would be working together

7. A "better than best-case" estimate happens when a programmer gives an overly optimistic estimate saying that there will be no problems whatsoever without taking into consideration life setbacks such as interruptions, having to rework different parts of development, or having unexpected problems. Realistically, there are always going to be

setbacks in software development, so it's better to not have a "better than best-case" estimate.

8. The best time to tell the customer that we will not be able to meet the delivery schedule is as soon as there are signs that show that the schedule is at risk since it will be possible to make the best amount of changes to have a successful project in the end. By communicating to them early, there are less surprises for everyone involved, and the customer would be able to adjust their expectations. While it may be a difficult conversation, as letting someone know that what they want isn't possible is always hard, it's necessary to be upfront and direct. To make it less difficult though, there should be progress shown and solutions offered to make sure that something promising can still come out in the end.

9. We think branching in software configuration can be good if managed well since it allows different team members to work on new features or fix bugs without making more problems for the other teammates. However, too many branches without merging can be dangerous since it can create a difficult merge later on. While developing our game Akemi Isles, we're working on separate branches that have different components being built that need to work together later on. While we have to develop the game manager, inventory system, and dialogue system separately, as it is in its design that they need to work by themselves first and foremost, there will come a time in the next few weeks where we will have to implement all these systems together. With this comes risk as it may not work out, but we're going to have to make it work in the end regardless of what problems occur. We think that what's most important though is that the systems we're building work and do their job no matter what as we can't merge the systems if one of them doesn't work at all.

10. We've been using build tools while developing Akemi Isles in Unity. The main ones we use are Unity's built-in build system along with Visual Studio's build tools for compiling C# crips. Unity's build system makes it easy to test a working version of the game on different platforms so that we can test out how the game will run for different users. Additionally, Visual Studio points out compile errors that need to be fixed before running the game. One of the downsides is that while the game may work in the Unity editor, there will be bugs that happen when testing out an actual build of the game which makes it frustrating since it can be impossible to know why these glitches happen exactly and make a whole mess of the project. Overall though, the build tools are extremely useful, and we can only be grateful to have them to make our project.