Article Summary 2

Problem 1

The two major concerns of any software project are quality and time, as the quality and schedule are the two driving forces. I feel as if neither is more important than the other as the two are essential to be planned out well together, that is, it is necessary to have both a good schedule and a high quality product. If you have a poor/tight schedule, your output product will most likely not be as high quality as you hoped, yet most of the time the only way to achieve a high quality product is to have a more lenient/good schedule, so the two tend to go hand in hand. The idea of complete functionality I feel is essential to the two and perhaps even a step lower in the foundation of software engineering. An incomplete or non functional project is flat out unusable, no matter the quality or the schedule set out to build it, so that is, this quality of completely functional fits in between the two, perhaps even being more important than both schedule and quality.

Problem 2

In the agile method for software development, the five main phases which typically occur in each and every iteration are planning/defining, build, testing, review and retrospective. The original project planning/defining that is done at the start of the cycle can get repetitive if done every single cycle, especially if the overarching plan/design is unchanging and therefore is the step that can be removed, however the main component which seems to become repetitive in the project comes in the last few steps where we test, review, and have a retrospective, all steps which seem to be repetitive as the retrospective comes after we have already reviewed the project, it seems as if the agile method lingers too long. The most optimal change would be that we remove one of these last phases or perhaps incorporate it into a different phase, as the retrospective and review can perhaps both fit into one.

Problem 3

The waterfall methods were heavily influenced by the practices of construction project management – the mantra was “find ways to build software like civil engineers build bridges.” (Essence of Software Engineering) These projects typically are very linear and the methods do not tend to overlap or get repetitive in the same way agile methods do. The main steps in a waterfall project are 5 non repetitive methods. The first one is requirements analysis, where all the projects documents/requirements are gathered and documented. Then system design, where the system architecture and overarching design is chosen. Next implement the code or build the project in mind. After this we have a testing phase where the built project is extensively tested. Lastly there is maintenance, where the project is ensured to last and maintain itself for future reliability. This methodology of project completion differs from agile in the repetitive review of the task after it has already been completed. Agile seems to waste time on users reviewing tasks over and over and even giving a retrospective of the task at hand, time wasted here where waterfall instead uses a more linear methodology which jumps into the next task at hand. Agile also seems to instead deliver lots of small tasks while waterfall instead only has one main deliverable at the end of the project.

Problem 4

A user story is a user written description of a general feature. Blueskying is to introduce ideas that are not restricted by imagination, essentially as free as the blue sky. User stories should describe one thing that the software needs to do for the customer, be written using language that the customer understands, be written by the customer, and be short (aim for no more than three sentences). User stories should not be a long essay, should not use technical terms that are unfamiliar to the customer, and should not mention specific technologies. The waterfall method does not typically have user stories.

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.

While I contest this claim, I argue that not all assumptions are necessarily bad. Some are necessary for moving forward projects, especially when information is limited. At the earliest opportunity, you should confirm assumptions. Unverified assumptions can complicate things but appropriately informed and based on expertise assumptions can lead decision making. Assumptions, therefore, are not necessarily bad, but should be used with care and evaluated shortly after there is a new piece of information.

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

I agree with this statement because larger user stories are too complex and have too many variables to do accurate estimates. Smaller clearly defined tasks can be easily decomposed from large user stories, providing more accuracy of estimation, prioritization, and tracking and in turn minimizes risk and uncertainty. In agile development, smaller stories lead to more predictable outcomes, better progress tracking, and flexibility to adapt as the project evolves, which is why big user stories tend to be problematic for planning.

Problem 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: Blueskying, Observation

I help you get more from the customer: Role playing, 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

NOTE: when you have finished, check your answers with the result in your text on page 62. Do you agree with the book answers? If you disagree with any of them, justify your preferred answer.

Problem 7

A better than best-case estimate is an overly idealized time frame that a developer gives when they assume the best possible outcome, but in doing so, they overlook potential risks, errors, and the involvement of others.

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?

If you discover there is a realistic risk you'll miss the delivery schedule, then it's best to tell your customer immediately. By communicating early, the customer can alter their expectations, make new plans, or even alter the order of the project based on the issues at hand. If we wait too long, it can damage trust and put our customers in a state of being blindsided. It is important to do so because it is evidence of taking responsibility and respecting their time, and opens the door to collaborative problem-solving.

Yes, it can be a difficult conversation, as no one likes to deliver bad news, especially when expectations are high. To make it less difficult, you could prepare by presenting alternative solutions, such as proposing new timelines, offering to deliver part of the project on time, or outlining the steps you'll take to get back on track. Focusing on transparency, accountability, and problem-solving can help maintain trust and poster a productive discussion.

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 software configuration, but I think it can easily trip you up if you don't know how to use it properly. Parallel development implies that developers can work in isolation from one another on features, fixes, or experiments; branching is totally nonblocking and such development is fully possible. Plus, this is especially helpful when working in a team on different parts of a project, in this case my group's current web software project based on React + Vite. It empowers us to make experiments or change, and not affect the core functionality.

But used poorly, branching can cause 'merge conflicts', where a couple of developers have changed the same files but in a conflicting fashion. Our project is a good example, if we are both working on frontend while another one is trying to integrate some backend things in separate branches, not doing merges often with main branch can lead to some very complex configs. In order to combat this, it’s important to stay on top of regular updating of branches with the main codebase and have a clear channel of communication with the team.

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?

For our React project, we’ll be using Vite as a build tool in our development. Vite is a lighter package that results in faster startup and Hot Module Replacement (HMR), which means that the page refreshes right away when code changes, without performing a full refresh. In turn this helps us iterate quickly, especially when making changes to the UI.

The downside, however, is that sometimes, Vite isn't perfect, it doesn’t play nice with some older libraries or tools that are traditionally supposed to be used with Webpack. While this can avoid some extra overhead in case of integrating more complex or legacy systems into our modern stack. All that said, the simplicity and speed of Vite far outweigh the disadvantages for our project setup now.