Pilone Reading Assignment 🚀

Lucian Prinz, David Yang, Davis Banks

Problem 1

The two main concerns of any software project are: how much will it cost, and how long will it take? While both are constraints, I feel like the bigger concern is time. Shortening the time a project takes can also decrease the amount that your resources cost, and the biggest resource is your development team, who's being paid for their time. The amount you spend on your resources can be a factor in allowing your software's functionality to be complete. Not having enough time to complete a project can also prevent a project from reaching complete functionality.

Problem 2

The 5 main phases that occur in every iteration are: 1. Planning, 2. Designing, 3. Coding, 4. Testing, and 5. Reviewing. One step that I think could be done once at the start of the project instead of being repeated every iteration is high level planning, because once a roadmap is set out, phases 2-5 can be performed without needing to plan each component of the project. Although unexpected issues will come up, which will need to be addressed within iterations, I believe that this will save time overall, even though additional time will be necessary in the beginning to plan out the entire project.

Problem 3

In the Waterfall method for software development, the main phases that occur are 1. Requirements, 2. Design, 3. Implementation, 4. Verification or testing, 5. Deployment & maintenance. Compared to the agile method, the waterfall method is linear, with each phase needing to be completed before moving on to the next. The waterfall method requires extensive documentation and planning beforehand, making it difficult and expensive to make changes once a phase is complete. In contrast, the agile method is done in quick short sprints, and requirements in the agile method can evolve based on feedback from stakeholders/customers. Overall, this makes the agile method a lot quicker. One example where you might need one of the Waterfall phases while using Agile is if you're making software for a regulated organization like the government, or making a medical device that needs to be cleared by the FDA. A team might be using agile, working in sprints and delivering features rapidly, however, they need to wait for

approval from the FDA or submit more detailed documentation before they continue which might slow down their work by a few weeks. Overall, I think that there are aspects from both waterfall and agile that are beneficial, resulting in trade offs between the two.

Problem 4

A user story is the perspective of the user, which usually describes a functionality or feature that they desire, and also describes the who, what and why of a requirement. Blueskying is a type of brainstorming where people think creatively and freely without thinking about limitations or constraints, which allows them to generate more innovative ideas.
A user story should clearly describe what the user wants, be focused on one functionality that they want without unnecessary details, provide a genuine benefit to the user, and be testable, so that the functionality can be verified.
A user story shouldn't specify the tech stack or implementation specifics, as this can limit the programmer from devising the best implementation approach, be too vague in what they want, and be concerned with stakeholder value, which can detract from the user experience.
The waterfall method does not use user story, as it is more of a sequential approach to developing software where a list of requirements are decided in the beginning and user feedback does not impact iterative development that much.

Problem 5

In software development, some assumptions are good. Assumptions should be made, such as the idea that a new feature should be accessibility compliant, but these assumptions should always be clearly documented. In a sense, they should rather be expected to be included, rather than assumed.

A big user story estimate might be a bad user story estimate because it could leave a lot of room for ambiguity in a design. It's simpler and clearer communication to make small changes, and clearly define the little things to prevent different assumptions from being made and provide consistency.

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: **Observation, Blueskying**
I help you get more from the customer: **Observation, Role Playing**
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 generally agree with the answers from the book. It could be argued that more observations can make things become more clear, but those end up being user stories.

Problem 7

A better than best-case estimate is an estimate commonly made by software engineers when answering a question about something, for example the scope of a project. In the book, it gives an example about if you ask a programmer how long something will take, they might say “No problem, I can do it in 2 days”, when in reality, they’re making a huge assumption that nothing will interfere with their progress, both in their life and in the code itself. A better than best-case estimate assumes there are no bugs, code works the first time, everything runs smoothly, their mom doesn’t need them for dinner, they don’t get sick, underestimates the difficulty of the program, runs into unexpected errors, etc…. I personally think this is a very common thing amongst software developers so I can relate to the book on this.

Problem 8

The best time to notify a customer that the delivery schedule is not able to be met is as soon as you become aware of the delay. I believe this is the best time because communication and transparency is key when working on behalf of somebody else. It will be a difficult conversation, but I feel that taking accountability for overestimating the scope of the project, being apologetic, and discussing potential solutions including offering alternatives or extending the timeline will ease the conversation and hopefully result in a more realistic project development schedule.

Problem 9

I actually believe that branching can be an incredibly safe way to keep software maintained. In an environment I’ve worked in, everybody was required to essentially work off of their own branches. Everybody synced with the main branch, but each person’s “branch” was compared every time a change needed to be merged in order to prevent possible instabilities existing in the main branch. Multiple production branches

existed as well, to ensure that the software was stable for a certain amount of time before being released to a mass audience.

Problem 10

I have used an internal build tool to compile a frontend project. The tool is proprietary and so the name can't be shared, but its function was to compile frontend code with flags set for features that should be enabled. This allowed quicker startup when working on features that didn't require the whole app to be loaded, also putting less stress on your development machine. I didn't like that it was always an extra step and was a little confusing to set up initially, but after a small learning curve it saved lots of time.