Talia Bahar
Lauren Lindsey
Shanaya Nagendran
Maya Pegler-Gordon
Week 9
Software Engineering Lab

Pilone & Miles Book Assignment

Problem 1

The two concerns to any software project are “how much will it cost?” and “how long will it take?” I think that the most important one is “how long will it take?”. This is important so that customers can remain interested in your product, and releasing a product as soon as possible gives you a competitive advantage over others. You can try to spend less money by creating a scrappy MVP of your ideal project, release it to people, and once it gets more attention or customers, it can be scaled up into a more costly project. This iterative approach means that the first version of the product won’t be completely functional, but will hopefully get there after a few releases.

Problem 2

The four main phases in each iteration cycle are gathering requirements, designing, coding, and testing. I think that they all need to be done in each iteration, because all of these aspects can change every few weeks. For example, you may have had a meeting with the customer and thought you knew what the requirements were, but once you built an MVP in iteration 1 and showed it to the customer, they realized that they want to change the requirements. This change in requirements trickles down to changing design, code, as well as how a product is tested. To avoid “big bang” software development, each of these steps must be done in each iteration.

Problem 3

Both Waterfall and Agile go over the same phases that iteration does (requirements, design, code, test). They both go over the same phases, but in the Waterfall method they only occur once in contrast to Agile development which will restart the processes every 2-3 weeks. The Waterfall procedure has a few additional steps such as risk assessment, prototyping, quality assurance, marketing, or finding an exit strategy. Agile projects might skip some of those phases within each iteration in order to shorten the development cycle. For example, an Agile team can create a product but might be missing an exit strategy until they absolutely need to create one.

Problem 4

- What is a user story?

    - A user story is “a story about how their users interact with the software you’re building”. They capture what it is a one is trying to build from the customer’s perspective.

- What is blueskying?
    - Blueskying is done when brainstorming solutions to the problem at hand where all ideas and solutions are suggested and none are ruled out. This is called blueskying because the “sky's the limit”.
- What are four things that user stories SHOULD do?
    - A user story 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 no longer than 3 sentences.
- What are three things that user stories SHOULD NOT do?
    - User stories should not be very long-winded, use terms that the customer would not understand, nor should it mention specific technology.

Problem 5

- All assumptions are bad, and no assumption is a good assumption.
    - Not all assumptions are bad, but to use assumptions correctly to gather estimates, teams should get the input of multiple people as well as rely on outside sources to get these assumptions. A bad assumption would be an assumption that is not based in any sort of logic or experience, because that can cause problems with roadmapping and estimating a project's deadline.
- A big user story estimate is a bad user story estimate.
    - Agree - this means that the story is best broken into smaller user stories, or the team should talk to the customer to reframe it. A user story should outline a task that can be completed within one iteration of development thus, a big user story is not a proper user story.

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
- I help you get more from the customer: 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

I agree with these answers and their matchings, however the one I am a little hesitant about is #2: “The more of me there are, the clearer things become: User Story”. I feel like although more than 1 user story is beneficial, there should be some sort of upper limit when it comes to the number of user stories. For example, if there were over 100 user stories that were all unique, a product can get feature fatigue, or the main vision and goal of the product can get lost. But if there are multiple user stories that all speak along the same 5-6 use cases, then I believe multiple user stories are fine.

Problem 7

A better than best-case estimate is one that does not take into account pitfalls and outside forces. Many developers make unrealistic assumptions about their ability to complete tasks and therefore, provide timeline estimates that are much less realistic than they realize. This leads to estimates being shorter than what is feasible.

Problem 8

The best time to tell the customer you will not be able to meet their delivery schedule is as soon as possible. This allows the customer to come to terms with what their new product is, as well as adjust any marketing or advertising towards what is going to be ready by the deadline. It is also important to provide a revised schedule for the customer to manage expectations. When it comes to customers, it is best to be clear, open and honest. I think it reflects poorly on yourself as well as your team/company if you tell the customer at the last minute something won’t be ready. Also, if you tell them early and your team is actually able to finish the project within the deadline, the customer will be happy. It definitely is a difficult conversation, but as long as the team communicates that the work will get done eventually, just not within that time frame, the customer hopefully will be understanding.

Problem 9

Branching in software development is good! Branching allows software developers to keep making new changes to a version 2.0 of a product while version 1.0 is being used by customers.

A scenario where a team does not use branching: a team develops an app, it goes out as a Beta version that has 10,000 weekly users and is growing. They start to want to add in more features, software developer 1 decides to add a new animation to the navigation menu, and it looks good on their end, and they push it to the working repository. It worked on their Chrome browser, but is incompatible with the Safari browser so that the 10% of Safari users on their website were unable to navigate through the web page.

Problem 10

I haven't used a build tool in development, but I have used IDE's like XCode that have a "build" button that automates the build process. Benefits of this are that it is easy when you are first starting out to get your project to run, but a bad point is that you won't know how to run and build your project if you were to run the same code not using that Build tool.