Jeremy Goldberg, Alvin Lai, David Sackler, Andrey Varakin
10/25/20

Assignment 2: Peloni & Miles Book Problems

1. The two main concerns of software projects deal with delivering the product within the allocated time and staying within the allocated budget. We believe that delivering a product in time should be prioritized over staying in budget for a couple of reasons, one being that it will not cause backups in the product chain (assuming that the deliverable meets the minimum needs and requirements). Another reason is that it will give clients an actual finished prototype or product to assess for future improvement. Going over budget (within reason) shouldn't be as big of a concern since the product could potentially make up for it when released. The idea of complete functionality fits both of these concerns. Making a product iteratively gives clients to assess working products and allows developers to accurately determine what must be done. This helps to reduce complexity to get the product done on time and prevents the allocation of money towards irrelevant parts of the project.

2. Requirements, design, code, and test are the four main phases of Agile development. We think the requirement and design phases could be intertwined because we found it helpful to figure out requirements after creating a mental model of Synth Trainer. Seeing both at the same time helped us better understand some of the challenges we would face and what needed to be prioritized. Doing this would have saved us some time and the confusion in the beginning since we did not truly know what we were trying to build.

3. Requirements, product design, system design/architecture, coding, testing, and maintenance are the main phases of the Waterfall development. It differs from the Agile method in that it performs each phase sequentially rather than iteratively cycling through the entire process. With Waterfall, hiccups in any later phase travel upstream and back to the requirements stage, making the method expensive in terms of time and resources. Maintenance is left out with Agile but we feel that the nature of Agile development takes care of maintenance. Its addition could however be useful once a product has been released so that clients have a product that is up to date.

4. Write one-sentence answers to the following questions:
    a. What is a "user story"?
        i. A description about features of a software product written from the perspective of a user/client.
    b. What is "blueskying"?
        i. It is a way to gather product requirements by allowing clients to spew out any ideas that come to mind (no matter how ridiculous) and to encourage collective brainstorming.
    c. What are 4 things that user stories SHOULD do?
        i. Describe an individual action that the product needs to do for the customer
        ii. Be written with language easily understood by the customer
        iii. Be mostly written by the customer

        iv. Be concise (less than 3 sentences)
    d. What are 3 things that user stories SHOULD NOT do?
        i. Be an essay
        ii. Mention specific technologies
        iii. Use technical jargon that the customer doesn't understand

5. What is your opinion on the following statements, and why do you feel that way:
    a. "All assumptions are bad, and no assumption is a 'good' assumption."
        i. We don't think assumptions are necessarily bad, but they aren't necessarily good either. It is a good starting point for developers to draft some mental models of what a customer wants, but assumptions should always be clarified in our opinion. Simply asking questions or working through the design of a product instead of going off of assumptions will prevent many future headaches.
    b. "A 'big' user story estimate is a 'bad' user story estimate."
        i. We disagree with the statement because we believe a big user story estimate may help developers more accurately determine the time needed to develop a product. Leaving out details to keep a smaller user story estimate may eventually present problems because the develops may have created an unrealistic prediction of the time it'd take to complete a project. In that case, customers will not be very happy.
6. Fill in the blanks:
    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**
    c. I help you capture EVERYTHING: **blueskying, observation**
    d. I help you get more from the customer: **blueskying, 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**

    We think the answers from the book are logical and reasonable, our answers didn't deviate from theirs surprisingly.

7. Best-case estimates are concerned with the lowest possible time and cost required to produce a product. They take into account all possible hiccups in the production chain to yield an estimate of what it'd take to minimize time and cost while maximizing satisfaction and revenue. A better-than-best-case exceeds the expectations of the best-case estimate and yields results better than the best-case estimates.

8. Once a team has exhausted their resources and concluded that they will not be able to meet a deadline, it is only appropriate to tell the customer immediately. We believe transparency with the customer as soon as possible prevents a much more difficult conversation later on if the problem was not brought up. Revealing issues early on may allow the customer and developers to work together and come up with a new deadline, prioritize requirements, and so on. Bring up problems immediately may also strengthen the customer's trust in the developers since it shows great respect and communication.

9. We personally stand by branching to avoid potentially breaking and blasting good, working code into oblivion. It allows for multiple developers to make the changes they need while keeping the integrity of the original code. While working on Synth Trainer, some of us were worried that we would make changes that would completely break existing software. Because of that, we created a new branch to experiment with and add our own changes to. This lets us feel at ease that some of our mistakes wouldn't cause errors anywhere else. We have experimented with Tone JS and certain React components that we are unsure of. These can later be reviewed by the rest of the team and merged to master if suitable.

10. For Synth Trainer, we are currently using Node JS. It's a powerful tool that allows us to use other extremely helpful libraries and frameworks such as React and Tone JS. A great advantage is that we can develop our projects mostly using JS, and the only disadvantage so far is its learning curve for those who haven't used it before or for those who've only used it a few times before.