Katie Nguyen

10/30/19

CMSI 401: Pilone & Miles Book Assignment

**Problem 1:**

The two major concerns of any software project consist of how much it will cost and how long it will take. Personally, I feel that the cost of a project is more important. This belief is due to the fact that while time is extremely important in regards to satisfying customers, timelines can be extended. On the other hand, when a project needs more funding, acquiring more money is not always an option. In other words, while time and money are both valuable, it is often more the case that time is easier to obtain. Complete functionality fits within these concerns because software is not complete until it is released with all necessary requirements, and time and money constraints all factor into release dates.

**Problem 2:**

In the Agile method of software development, the four main phases that occur in each iteration consist of identifying requirements, designing, coding, and testing. Out of all the phases, I feel that designing could possibly be done at the start of the project, rather than in every iteration. A redesign might be necessary as the customer provides feedback throughout the project, but I do not foresee a complete redesign being necessary. Overall, I do not believe that this would save significant time on the project, but rather prevent wasted time or inconveniences for the developers.

**Problem 3:**

The main phases that occur in the Waterfall method for software development consist of requirements, design, implementation, verification, and maintenance. In contrast to the Agile method, the Waterfall method consists of an implementation, verification, and maintenance phase in place of a coding and testing one. In regards to the structure of the waterfall method, I do believe that the extra phases are needed to complete the linear structure of the Waterfall method. Consider the scenario in which a team was using the Agile method to work on an app in sprints. If one of the packages that the app depends on becomes outdated in the next sprint, the added maintenance phase in the Waterfall development process might be needed in order to keep the project running.

**Problem 4:**

* What is a user story?
  + It is a story about how users interact with the software that is being built.
* What is blueskying?
  + Blueskying is when you iterate with the customer on their requirements.
* What are four things that user stories SHOULD do?
  + User stories should describe one thing that the software needs to do for the customer, are written in a language that the customer understands, are written by the customer, and are short.
* What are three things that user stories SHOULD NOT do?
  + User stories should not be a long essay, use technical terms that are unfamiliar to the customer, or mention specific technologies.

**Problem 5:**

I mostly agree with the statement that “all assumptions are bad, and no assumption is a ‘good’ assumption.” Assuming a customer’s needs and wants is never a good idea, as it can lead to outcomes that the customer did not want. In this sense, no assumption is a good assumption. On the other hand, assuming a trivial piece of information, rather than bothering the customer with every small detail, might prove to be more beneficial in the long run.

I completely agree with the notion that “a ‘big’ user story estimate is a ‘bad’ user story estimate.” I believe this idea to be true because a big user story can often result in an overly detailed synopsis that is incapable of being built. Too much detail can also lead the customer to have a narrowed version of what the end product will be, rather than being open minded to what it can become.

**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
* 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 the book answers. The only difference between the two is that the book also lists Observation in addition to Role Playing for “I help you get more from the customer.” However, I can see how both answers are applicable to this statement.

**Problem 7:**

A better than best-case estimate is the response that a programmer will give as to how long it takes to get something done, such as writing an interface to a database. For example, programmers will say that they can get a task done in two days. However, in this statement, they assume that they are the only people involved, no mistakes will be made, and testing does not have to occur. In reality, the thought process behind this timeframe is that if the programmer stays up all night they can crank it out, under the assumption that nothing else goes wrong or comes up, which is all far from controllable.

**Problem 8:**

In my opinion, there is no great way to inform a customer that you will not be able to meet their delivery schedule. However, if this must be done, I believe that it would be best to deliver this news at the beginning of the project, or at the earliest moment into the project when the developer realizes that the due date cannot be met. In this regard, the customer will be informed as early as possible and have sufficient time to adjust as needed. I do believe that this would be a difficult conversation. However, being upfront and honest with the customer will ultimately have the greatest payoff in the long run. The customer might initially be upset, but after the information has been delivered and they have time to process the outcome, they will be able to change their expectations and deliverables on their end. Additionally, it is my belief that most customers would rather have a completely functional, quality product that takes longer than expected, rather than a piece of software that is on time but is not what they wanted, or worse, does not work properly.

**Problem 9:**

I believe that branching in a software configuration is always a good idea. In this sense, developers can make changes and progress through their project with the security that they have a working version of their code. It is easy to break what was once working due to a typo, bad decision from a co-worker, or crashed hard drive. However, with the version control that comes from branching, it can be guaranteed to some degree that there is a “safe” version of the code somewhere in a repo. This allows developers to undo mistakes in recent versions, and make necessary bug fixes to both new and old versions of the team’s software. For instance, if a team member were to change the type of a variable without consulting the rest of the team, and that type change broke the code, then the team could reference older versions of the code to identify what caused the code to break and undo changes as necessary.

**Problem 10:**

In regards to my development project in 401, we have used simple scripts to maintain the pipeline for retraining our neural network. These script commands do a good job of kicking off the commands that we need from anywhere in the script. However, on the flip side, scripts do not execute correctly unless the order of the commands are based on the dependencies as they are needed in the path of execution. In this regard, if script commands are executed in the wrong order they fail in largely unidentifiable ways.