Parker Hague

Dean Relland

Assignment: Assignment 3 Extreme Programing

Date:10/1/21

Part 1:

Pair programming was an interesting experience. It felt weird to program and have someone watch over my shoulder. I feel like at first, I was making more mistakes just because I was distracted by the fact that someone else was watching me. It was nice and constructive to be able to program together with another student. I found it interesting getting to see another person’s programming style and comparing it to my own. The advantages of pair programming were that Dean, and I were able to talk about the implementation and we were able to help each other out when we ran into silly little errors. The disadvantages I can think of are that it could slow down the programming process if something simple was being programmed. Also, if you’re distracted by someone looking over your shoulder then it could possibly slow down your programming speed. Ultimately, pair programming was a new and good experience that I enjoyed.

Part 2:

**Pair programming:**

Advantages: Can reduce programming errors and the time it takes to find an error. Can also improve understanding of the code when talking about the project with another person.

Disadvantages: Code production can potentially be slower because the second person could be doing something else. Costs more money to have two developers working on the same thing at the same time. Not as beneficial on simple projects.

**Metaphor:**

Advantages: Metaphor give a unified labels to parts within the programming process. It’s main advantage is better communication.

Disadvantages: Metaphor can be hard for some to understand. Some people in the industry don’t like extreme programming.

**Refactoring:**

Advantages: Refactoring can give one time to go through code and make sure it’s as optimized as possible. It can also help reduce code clutter and help the revision process go smoother.

Disadvantages: Refactoring means time and money has to be spent on code that’s already written and working. This can make refactoring more costly. Sometimes code doesn’t need to be refactored because the current implementation is good enough.

**Timebox:**

Advantages: Clear specified time to work can make developers more efficient. Also, allows development process to be adjusted on the fly do to the analysis at the end of timeboxes.

Disadvantages: The overplanning can possibly slow down simpler projects. Too many timeboxes can end up slowing down developers instead of being beneficial.

**Iteration:**

Advantages: These allow programming to be done in smaller segments and then adjusted as needed. These are beneficial when getting feedback from users and when a development project is constantly changing and evolving.

Disadvantages: Like the last one, too many iterations in a short amount of time will end up slowing down developers instead of helping them. Also, may not be necessary on smaller and simpler projects.

**Sprint:**

Advantages: A shorter period of work where many people can share a similar goal thus making development more efficient and also able to react to change.

Disadvantages: Sprints aren’t designed to work for every single project and development team. A sprint that’s too long can slow down a smaller project and too many short sprints can also slow down dev teams.

**Onsite customer:**

Advantages: Continuous feedback. Not having to stop development and wait for the customer approval. Can greatly speed up the development process for some apps and situations.

Disadvantages: Customers don’t always know what they want. Sometimes having to constantly get a customer’s opinion can really slow down the development process. Working in bigger chunks and then showing the customer can sometimes be better.

**Test Driven Development:**

Advantages: Having proof that a method does what it’s supposed to. Can greatly reduce bugs/errors. Easier to track down bugs. Can be very beneficial when working with large teams and large production applications.

Disadvantages: Can really slow down development process. It’s also a lot more code to write. A developer can forget to write the test or even perform the test. Typically not necessary on smaller or simpler apps.

**Unit test:**

Advantages: Reduction of bugs and easier to track down errors when they occur. Also allows the developer to work and automatically know if their code is still working or if it has broken.

Disadvantages: Can be cumbersome to write. Not always necessary on every function. The time taken to write the tests will definitely slow down the developer. Usually the project has to be somewhat big to see the benefits of unit tests.

**Test first approach:**

Advantages: Allows programmers to know the exact requirements for their function. This can make the actual implementations simpler and allow the programmer to understand their task better. Another advantage is getting immediate feedback when finally doing the actual implementation of the function.

Disadvantages: Doesn’t work in every situation. Sometimes developers don’t fully know how the function should be tested until after it has been written. Can also be hard to do with more complex functions.