Chapter 4. Automate Your Coding Standard.

**Three Things I learned today**

1. **Before** - I didn’t know that code formatting would be important for a team and is part of a coding standard programmers have to follow.

**After** - Now I know that it’s actually a good practice to follow a well formatted code standard and that every team of programmers need to set and follow.

1. **Before** - I never fully knew the reason why programmers have to follow coding standard if it’s such a hassle, I usually just write down code without any format to follow since it’s my own code and I understood it well.

**After** - It’s not actually the right way of doing things and programmers follow these coding standard in order to make their codes uniform and understandable to others working on it, this way it’ll be easier and faster to work in the project.

1. **Before** - As I understand everyone has their own way of writing code that suits well and easier for themselves to work with, but when they work with others it’s not usually the best practice.

**After** - There exist tools that can be used to help programmers to produce quality code and maintain coding standards and formatting which is automated and should be included in the build process as well as being dynamic instead than static.

Chapter 5. Beauty is in Simplicity.

**Three Things I learned today**

1. **Before** - I didn’t think that the simpler the code the better, I assume that when codes are more intricate and complicated the more functionality it has to offer and more detailed and versatile it is to the program.

**After** - it’s not usually the case that the more complicated it is the more function it can perform, sometimes the simpler the code the better it is to be understood, maintained and improved by others.

1. **Before** - i didn’t know that developers also follows value the they should aspire to, I thought that its only about communicating to other programmers about the code and projects to produce products fit for what the clients want.

**After** - Now I know that programmers should also care about the code they produce and that it should have readability, maintainability, speed of development and the quality of the code.

1. **Before** - As I understand, programmers have each of their ways they perceive how their code works and how they approach to write these codes depends on the habits developed overtime.

**After** - programmers adapts these codes they use from other source codes they studied and practiced which they combined and created their own signature of writing codes which in their own perception is what they perceived as beautiful and simple.

Chapter 6. Before You Refactor

**Three Things I learned today**

1. **Before** - Refactoring as I understand it is to change someone’s else’s code if it’s not up to the company or clients standards or if its way behind and is old and needs to be replaced with a new improvised way of coding standard but it’s not actually the case.

**After** - There’s actually a lot to consider that goes into refactoring an existing code before one should start changing it and I learn multiple approach on how to handle code refactoring.

1. **Before** - When I think of refactoring, the approach I follow is usually just basing it on if the code has some error that needs to be fixed and if the code can still be improved in a way that will add more functionality that is beneficial to the overall performance of the system.

**After** - I found out that the best way to approach refactoring is by taking notes of the existing codebase and the test cases of that code, this will help the programmer to understand more about the process and the function of that code, which will lessen the need to rewrite everything and reuse only the strong points of that code, it also helps to change the code little by little in order to measure the impact it does on the overall system performance.

1. **Before** - Refactoring does not necessarily proves that the old code is not functioning as its require to do, but sometimes it does not fit the new design and standard of the overall system anymore and needs to be replaces as is without having to test if it’s still applicable to be reused.

**After** - I realized that it’s important to test the old code after each iteration of changes being made in order to measure the old codes functionality and how much of old code can be reused and why those tests that failed are the way they are. It also important to note that personal preferences and biased shouldn’t be applied when refactoring a code only because you don’t like the old code or that you think the new improved function or framework you learned fits best with the system without testing if it’s really beneficial, sometimes the old code performs just as better than the new one.

Chapter 7. Beware the Share

**Three Things I learned today**

1. **Before** - I feel the same with what the chapter said, reusing codes I’ve written from the past and previous existing lines of codes or projects that would have the same function as one another and that in my view also fits in the function of the current feature I am working on.

**After** - I realized that when you reuse existing shared codes those functions that are independent on one another can now become dependent in a way that could create an error if left unnoticed, this can be caused by the same variable being used from the same line of codes that was reused, or due to the same logic that the different function performed which created dependencies on each other.

1. **Before** - I thought that reusing existing codes seem like a good idea to save time and resources when creating a project and has no effect with one another as long as the system performs as it should.

**After** - but in the long run it actually creates small problems that are unnoticeable at first but overtime it accumulates harmful effects that damages the performance of the overall system. Its more advantageous and saves more time each function for each feature are made without reusing existing shared codes as it minimizes the problem or errors and contain it for easier fixing and debugging.

1. **Before** - I usually reused existing codes to those features that has somewhat the same functionality and produces the same result, without much thought of what the difference between each function is or if it has any effects to those existing shared code and what it can do to the overall process of the system.

**After** - It’s important to note the missing critical thing with reusing the existing shared codes and that is the context on how the code should be reused when or why those lines of codes exist and if they somewhat correlates to each other or acts independently with one another, programmers should put into consideration if the features that has somewhat the same functionality are dependent on each other or not and if they can evolve independently with one another.

Chapter 8. The Boy Scout Rule

**Three Things I learned today**

1. Before - I never experienced working on a proper team, when creating our capstone project, it only consisted of two coders that have each different task with one another but still the overall process of the system correlates with one another so it doesn’t relates to me much on the importance of having to clean and improve other peoples code other than my own.

After - I do see the importance of having to contribute and improve on the module or system especially when working as a team as it gradually makes the code cleaner and better than when every other people only focuses on their own assigned tasked and module of the system.

1. Before - I only focuses on improving and cleaning my own code before as I did not have the chance to work well with others and I see that it’s not a desirable trait to as a programmer that typically works as a team.

After - I know now the importance of contributing and helping each other as part of team and not as an individual, this way more features can be done faster and easier if everyone is gradually contributing to make the system better and better

1. Before - I thought that a team which has each task individually assigned to them would only need to perform those task given to them and would have to adapt and structure their code depending on the code their team made.

After - it’s not actually about individuality of the team but about helping one another acting and caring for the system as a whole, in order to operate and make the code better than before.

Chapter 6. Before Your Refactor

**Three Things I learned today**

1. Before -

After -

1. Before -

After -

1. Before -

After -