**97 Things Every Programmer Should Know**

**Chapter 14: Code Reviews**

**What are Three Things I learned today**

1. Before – **I have a short amount of time reviewing my code.**

After – **Code reviews increases code quality and reduce defect rates so it should be given an amount of time.**

* Reviewing of code consumes time indeed, but it contributes great amount of quality and reduces defects in the process of coding.

2. Before – **I haven't tried any code reviews, so i have no idea how crucial it is in coding.**

After – **I realized that code reviews holds essential part in creating a quality of code.**

- In my experience before, I never consider code review as part of our coding as long as it runs, I am good to go but now that I realized its importance, I will apply it to my future projects.

3. Before – **Code reviews are simply just creating changes and correcting mistakes.**

After – **Code reviews primary purpose is to share knowledge and establish common coding guidelines.**

* Instead of simply just looking for errors one should review the code by trying to learn it and understand it and by that, it will produce better outcome.

**Chapter 15: Coding with Reason**

**What are Three Things I learned today**

1. Before – **Trying to reason about software correctness by hand results in a formal proof that is longer than the code and is more likely to contain errors than the code.**

After – **Creating sections for reasoning helps the programmer understand and can easily identify the intent of each code. It also simplifies the reasoning to make things easier.**

* Dividing sections according to their respective purpose helps the programmer trace and identify the code without having confusion.

2. Before – **I don't have any idea of any coding practices that will help improve my code.**

After – **There are practices that I should consider to produce quality code and will improve my coding skills. These are:**

* Avoid using goto statements, as they make remote sections highly interdependent.
* Avoid using modifiable global variables, as they make all sections that use them dependent.
* Each variable should have the smallest possible scope.
* Make objects immutable whenever relevant.
* Make the code readable by using spacing, both horizontal and vertical.
* Make the code self-documenting by choosing descriptive
* If you need a nested section, make it a function.
* Make your functions short and focused on a single task.
* Functions should have few parameters
* More generally, each unit of code, from a block to a library, should have a narrow interface.
* In order to preserve class invariants, usage of setters should be discouraged, as setters tend to allow invariants that govern an object's state to be broken.

3. Before – **I have no idea what creating arguments are for.**

After – **Creating arguments for reasoning about the correctness of the code helps you understand more about it.**

* Instead of focusing more on reasoning the software correctness in a proof that is longer than the code, I will try to divide it by section and create arguments accordingly to gain insights that will be helpful in the future, not just for myself but also for the team.