**97 THINGS EVERY PROGRAMMER SHOULD KNOW- CHAPTER 4: AUTOMATE YOUR CODING STANDARD**

**WHAT ARE THREE THINGS I LEARNED TODAY**

1. **Before**- My code is very messy especially when I create complex functions for various complex processes.
2. **After**- Code formatting is also important in development because it provides ease of navigation especially when debugging the code.
3. **Before**- I put low priority on using code analysis tool because I thought that they are just hassle especially when additional steps are required upon installing it.
4. **After**- Using static code analysis tools to scan code for unwanted anti-pattern is very essential, they contribute to the overall efficiency of the code.
5. **Before**- My coding standard before doesn’t have strong flexibility especially when injecting new changes for revisions of a specific module.
6. **After**- The coding standard should be dynamic rather than static as the information stated because an efficient code should be flexible and adoptable to various changes especially when developing a complex module.

**97 THINGS EVERY PROGRAMMER SHOULD KNOW- CHAPTER 5: BEAUTY IS IN SIMPLICITY**

**WHAT ARE THREE THINGS I LEARNED TODAY**

1. **Before**- I don’t care about the beauty of the code, what matters is to make it work and functional.
2. **After**- Simple code speaks for itself.
3. **Before**- Having a messy code is fine as long as it has comments and documentation
4. **After**- Even if the code has comments and documentation, if it’s really messy other people might still find it quite hard to read and understand.
5. **Before**- Beautiful code is very complicated code or in other term “Overengineered Code”
6. **After**- A beautiful code is a simple code that has simple responsibilities all throughout the system.

**97 THINGS EVERY PROGRAMMER SHOULD KNOW- CHAPTER 6: BEFORE YOU REFRACTOR**

**WHAT ARE THREE THINGS I LEARNED TODAY**

1. **Before**- I sometimes reengineered the code too much that instead of it solving problems, it adds up more problem that’s need fixing.
2. **After**- I should carefully examine the existing code and only refactor the part where it fails to do its job or responsibility.
3. **Before**- When a specific set of codes fail to work, I am always tempted to rewrite everything and start from the beginning.
4. **After**- I should reuse the functioning part of the existing code as much as possible.
5. **Before**- I tend to change the code from other programmers and let my ego take me over.
6. **After**- I should just leave the other programmer’s code unless they are so buggy that they need to be revised or reengineered.

**97 THINGS EVERY PROGRAMMER SHOULD KNOW- CHAPTER 7: BEWARE THE SHARE**

**WHAT ARE THREE THINGS I LEARNED TODAY**

1. **Before**-. Using a shared library code is always efficient and recommended
2. **After**- Shared library code should be used appropriately depending on the context.
3. **Before**-. Using a shared library code brought convenience to the developer in any situation
4. **After**- The shared library code should be examine thoroughly to determine if it would really fit to the existing code that is being developed.
5. **Before**- I wouldn’t mind checking the shared library code before using it into the development as long as it do its job.
6. **After**- Checking shared library code is essential in development so that it will prevent any major conflicts in the near future.

**97 THINGS EVERY PROGRAMMER SHOULD KNOW- CHAPTER 8: THE BOY SCOUT RULE**

**WHAT ARE THREE THINGS I LEARNED TODAY**

1. **Before**- I don’t care about the code of my fellow developers
2. **After**- I should look after the code of my fellow developers when I got the chance to review their work.
3. **Before**- I don’t exert effort in making the code of others cleaner because I believe they know what they are doing.
4. **After**- I should try my best to offer the best thing that I can do in order to make their code cleaner and better.
5. **Before**- I don’t test thoroughly the changes that I have implemented on my fellow developer’s code.
6. **After**- I should respect their code and don’t try to leave a change that can cause a catastrophe to other developer. I should prevent doing littering.