# **97 Things Every Programmer Should Know**

## **Chapter 4: Automate Your Coding Standard**

1. **Before -** Reading the documentation, and follow the following format or syntax of the program code.

**After -** It’s really true that once the program system is done, the code looks like a mess. Sometimes, the developers don’t leave some comments for better understanding of the algorithm or a snippet of the code.

1. **Before** - Each developer does have different pattern or format of coding that they follow.

**After** - This sentence “following a code standard can be quite a boring task if it isn’t automated” makes me laugh and interested, cause since then we follow a certain code standard so it makes look clean.

1. **Before** - We follow a certain code standard for uniformity and that other developers can easily adapt the format.

**After** - The whole project team should agree on the code standards and formatting they should follow. Not all aspects can be automated, and the coding standards should be dynamic rather than static, in case the project requires changes.

## **Chapter 5: Beauty Is in Simplicity**

1. **Before -** When we hear the word "beauty", it's often associated with physical appearance. However, beauty is subjective and attracts someone's eye differently. Within our industry, I'd say that a program code is beautiful if it is easy to understand and produces correct output.

**After -** Every industry has a different definition or perception of "beauty," such as art majors who view the beauty of software by comparing it to art. Regardless of these differing perceptions, it all comes down to the concept of "simplicity."

1. **Before** - When searching for a code or solution, I usually look for a shorter code that is understandable, and test it if it is working on my end, and have the correct output that I’ve been looking for.

**After** - Finding a simple code has benefits, including a better understanding of how it works. Understanding the code is crucial because if problems or errors arise, you can easily identify the defect or issue in the program code.

1. **Before** - Simplicity is the key.

**After** - I believe that the key to creating beautiful code is in its simplicity. When code is simple, it becomes more readable and easier to understand for other developers and programmers. This is important because if there are any issues or problems that arise, it is easier to diagnose and fix them. Additionally, code that is easy to understand can be maintained over time, is clean and organized, and can be adapted to new requirements as needed. Overall, striving for simplicity in code writing can lead to more beautiful, efficient, and effective software.

## **Chapter 6: Before you Refactor**

1. **Before -** In refactoring a program code, I usually replace the old code with a new one.

**After -** I do agree that we should reuse code as much as possible and avoid rewriting everything, as it is time-consuming. Additionally, old code has already been tested and produces correct outputs. If we were to rewrite everything, it could result in numerous bugs and errors, slowing down the progress of the project.

1. **Before** - If there are some changes, I make sure to have it one by one, not a massive one. In this way, I have assurance that each module that I’m working on is working correctly.

**After** - It’s true that many incremental changes are better than one massive change. This approach helps to ensure that each component I work on is functioning properly. By making changes one at a time, I can easily identify and fix any issues that arise, which helps to minimize potential disruptions to the overall project. Additionally, this allows me to confirm that each modification has the desired effect, before moving on to the next change. Making changes in a controlled and methodical manner helps to ensure the overall success of the project.

1. **Before** - When working on our capstone project, before adding a new module to the repository, we ensure that the existing code is functioning correctly and then verify that the new module serves its intended purpose.

**After** - Before adding any new module to the existing system, it is crucial to first test the current system and make sure it is functioning correctly. By doing so, any issues that may arise after the new module is added can be traced to the new module, allowing for a more efficient and effective problem-solving process. This step is essential in ensuring that the program system remains stable and running smoothly, and can save time by avoiding the need for extensive debugging and troubleshooting.

## **Chapter 7: Beware the Share**

1. **Before -** Sometimes, I do just copy the available code, check if its working, then paste it to the project system.

**After -** Pulling out shared code into libraries without understanding the shared code could result to a mistake.

1. **Before** - After pulling out shared code and checking if its working, I don’t mind if it doesn’t meet the pattern or framework that the project system is utilizing as long it is working correctly.

**After** - When extracting shared code into libraries, it's important to ensure that the pattern and framework of the library code match that of your program code. This helps to maintain consistency and reduces the likelihood of errors, making the integration process smoother and more efficient. y following this practice, we can ensure that the code is easier to maintain and improve.

1. **Before** - Extracting shared code into libraries is the easy way to complete the project system.

**After** - Don't blindly extract shared code as it may seem like a good idea, as it could result in a greater cost than its worth to the project system. Simply extracting shared code for the sake of it may result in greater expenses for the project system and may not actually bring value to the system. As said, check the context before you proceed.

## **Chapter 8: The Boy Scout Rule**

1. **Before -** I do agree with the Boy Scout Rule of 'Always leaving the campground cleaner than you found it', but there is always an exception. Some people may prefer a messy environment and cleaning it without their permission could lead to misunderstandings.

**After -** Still I do agree with the Boy Scout Rule. While the intention of the rule is good, it is important to be mindful of different perspectives and always consider the consequences before taking action.

1. **Before** - Make sure that the added program code is clean and is better than the old code, as its purpose is to improve the quality of the program code.

**After** - Before adding a code, check the old code if its efficient and doesn’t need cleaning. This is important because if the old code is already functioning efficiently and doesn't have any issues, there may not be a need to make any changes or add new code. This is important because if the old code is already functioning efficiently and doesn't have any issues, there may not be a need to make any changes or add new code.

1. **Before** - Personally, I don't like messy environments as they irritate my eyes. So, I make sure to clean up as much as I can, including leaving fast food restaurants in a cleaner state before leaving."

**After** - Before cleaning up code in our field, make sure to notify your team and have a backup of the program code. Having a backup will ensure that you can quickly revert to the previous state of the code, minimizing the impact of any issues and saving time in the resolution process.