▼ Title Of Book and Chapter

## **(Chapter 9) Check Your Code First before Looking to Blame Others**

▼ What are Three Things I learned today

1. **Before** – As I've done in the past, I have a project that I created and I run into a lot of issues because I'm copying and pasting code from another source, even if I know the code, doesn't work for my project, or the code isn't functioning to mine.

**After** - I’ve learned that it is important to understand the Code although it takes the time to thoroughly understand the code you wrote and how it's supposed to work. This will help you identify potential issues and avoid blaming others for mistakes you made.

2. **Before** - In the past, I have also run into problems with projects where I don't know how to fix them or why they occurred. So, I got in touch with my classmate and asked him what the issue was and for his advice on how to resolve it.

**After**  - I’ve learned that it is important to have a colleague review your code before blaming others for issues. They may spot something you missed and can give you constructive feedback.

3. **Before** – When I tested my project, I discovered that it contains a large number of mistakes, and I had no idea where to begin debugging it. So I made the decision to test the code and identify the code that causes errors when it goes online.

**After -** I’ve learned that it also important to write test cases to validate that your code is working as expected. This can help you catch problems early on and avoid blaming others for issues that you introduced.

▼ Title Of Book and Chapter

## **(Chapter 10) Choose Your Tools with Care**

▼ What are Three Things I learned today

1**. Before** – I’ve experienced that I choose and choose tools what I will used but I didn’t get actually the requirements.

**After** - I’ve learned that it is important to clearly identify your requirements and know what the tools you select must provide. This will assist you in avoiding selecting tools that don't suit your requirements.

2. **Before** - I've found out through experience using various tools that some functions aren't supported when it becomes live.

**After** - I’ve learned also that the tools you select are compatible with one another, as well as with your current systems and procedures.

3. **Before** - On a project, I've used a variety of tools, and it's still functioning.

**After**  - I’ve learned that for future-proof I must take into account the tools' potential and make sure they will remain pertinent and helpful in the years to come. Keep abreast of the most recent advancements in the industry and select tools that will continue to fulfill your demands. Also, as the firm expands, I must be sure the technologies I use are scalable and can meet the demands.

▼ Title Of Book and Chapter

## **(Chapter 11)** **Code in the Language of the Domain**

▼ What are Three Things I learned today

1. **Before** – I've had the scenario before when I utilized words in my code that made it difficult for others to follow its logic.

**After** - According to what I've studied about coding in the domain's language, it's important to use words and phrases that are appropriate for the area in which the issue is being addressed.

2. **Before** – I've noticed that many people have trouble understanding the majority of my codes.

**After** - I also discovered that it encourages improved comprehension and communication between engineers and subject-matter specialists.

3. **Before** – As I've already mentioned, I've used jargon and convoluted code in addition to not utilizing domain terms.

**After** - I have discovered that the clearness and maintainability of code may be improved by incorporating domain terminology in class and function names. Be careful not to use programming-specific terminology or technical jargon that non-technical stakeholders might not understand.

▼ Title Of Book and Chapter

## **(Chapter 12) Code Is Design**

▼ What are Three Things I learned today

1. **Before** – I've had projects that I believed I could complete before the deadline turn out to be unsuccessful.

**After** - I’ve learned that the time line predictions will become less accurate with time. Because of the design cost of the project.

2. **Before** – I've previously worked on a project and failed to reach the deadline, producing work that wasn't very excellent.

**After**  - I've discovered that no project's output or result can ever properly fulfill its deadline on time, and the quality of the product can be as good.

3. **Before** – I've also had success in the past with "repairing the broken" on electronic devices or PCs. I became an expert in this field, and even my teachers referred to me as a "Tech Savvy Guy"; for this, I give thanks to God.

**After** - I've discovered that incredible designers devote their lives to perfecting their craft, producing excellent designs. Code is the same way.

▼ Title Of Book and Chapter

## **(Chapter 13)** **Code Layout Matters**

▼ What are Three Things I learned today

1. **Before** – I've had instances where I didn't even comprehend my own code.

**After** – I've discovered that maintaining consistency in the way code is laid out makes it easier to read and maintain.

2. **Before** – I've learned from previous experiences that I need to choose an indentation style (like spaces vs. tabs) and stick with it consistently. Using line breaks to separate logical pieces of code.

**After** – I’ve learned that a code’s layout is a part of expressiveness too.

3. **Before** – I know from experience that my own code is not formatted properly.

**After** – According to what I've read, whitespace should be used to separate the arguments in function calls and between operators in expressions. Using code formatting tools to automate and enforce a unified code layout (like linters), is a very big help as a programmer.