Chapter 9: Check Your Code First before Looking to Blame Others

**Three Things I learned today**

1. **Before** - When I encounter an error that’s been detected by the compiler I usually just test run the code and see if the error does exist or the compiler just made a mistake on detecting an error so if my code runs fine I just assume it’s the compilers fault.

**After** - I learned that it’s not always the compiler, Os, Interpreter, server, system software or the database that causes the problem, but our own code that causes those errors, we just have trouble believing it or have missed it.

1. **Before** - I once had a problem in my code where the compiler detected an error on the class function that is initiated and I keep thinking that maybe the error might be in the compiler or maybe I was missing a framework or an extension, I later found out that I didn’t use the library associated with that class.

**After** - Compiler bugs are rare and it’s a waste of time to prove and find the error that you believe is in the compiler, it’s much better to fix and find solutions on your own code and do a series of tests, search for every possibility within the code that can cause error, may it be from libraries, version number or variables mismatch and test it to other machines and try different configuration.

1. **Before** - I usually look for the error where it was specified, I did not first look at my own code and jump ahead to finding solutions to the problem where I thought might have caused the error not realizing that I was wrong and is wasting my time.

**After** - Look first on your own code and eliminate all possible errors it might cause and if it’s not there then whatever remains however impossible must be the cause of the problem as Sherlock Holmes’ advice.

Chapter 10: Choose Your Tools with Care

**Three Things I learned today**

1. **Before** - I had a problem of having no initial plan and didn’t research the needed tools before starting making the program and it caused us to have a problem later on.

**After** - I learned that it’s important to choose the tools needed to build the application carefully as it can affect the production of the program later on if not down well.

1. **Before** - The database we decided to use for the program had trouble connecting to the webserver when we deployed the application that caused us time and money.

**After** - Choosing the right mix of tools for building the program matters as well, if the tools that was initially decided upon might not be compatible with one another as the project grows in size and complexity.

1. **Before** - Deciding and using multiple tools from the start doesn’t seem like a bad idea at first, it would set the production and no need to worry about deciding what tools to use later on in the process but it’s not the case.

**After** - it’s better to use less tools and only the necessary at the start of the building process and then just add more external tools if needed as the project progresses.

Chapter 11: Code in the Language of the Domain

**Three Things I learned today**

1. **Before** - I usually just code in a manner that doesn’t put into consideration others that might read my code since I understood it well enough for my own.

**After** - It’s important to code in an understandable language that will be understood by others that would make working on the code base much more easily.

1. **Before** - I never thought that there are domain terms that would need to be followed which will make the intent of your code be clearer to others.

**After** - Programmers should code using domain terms and making the domain concepts explicit so that other programmers can gather the intent of the code much more easily

1. **Before** - I didn’t consider that it would be easier to change your code when you write it in the language of the domain as they continue to evolve.

**After** - writing code in the language of the domain terms means that when the domain model evolves the code will be in a good position to evolve as well.

Chapter 12: Code is Design

**Three Things I learned today**

1. **Before** - I never realized the importance of a good design in the code and that it can affect the whole project when not done correctly and if we only settle for a good enough design.

**After** - Blueprint for a construction project is like code, it’s the design that tells the machine on what to do in order to create the finished product, and those architects are the programmers that designs the code so they code are important and without its good design the product will collapse.

1. **Before** - Paying programmer’s high wages just for sitting and looking at their computers seem confusing, why is that? It’s actually because making the design is the most important part when creating something so that there will be minimal mistakes in the production process.

**After** - Design cost are more important than the construction cost, without a good code design, the timelines in finishing the product will become unreliable and may result in an incomplete project output that would cost more to rebuilt.

1. **Before** - I didn’t fully understand that coding is not a mechanical process and that it a creative mind in order to produce quality results.

**After** - Code itself is the design in which we make a creative process, it demands quality and it exceeds our capacity to build them. The design would need to be validated through testing until it can be called as complete.

Chapter 13: Code Layout Matters

**Three Things I learned today**

1. **Before** - I never realized the importance of a right layout when it comes to understanding the code I usually thought that the automatic formatter would take care of the format and layout.

**After** - the importance of a right layout actually has an effect on how programmers understand the code and it actually has advantages with the right format.

1. **Before** - I never realized the underlying reasons why formatting a code is important and how it affects the programmers mind of thinking and understanding.

**After** - I understand now that people have a natural good visual recognition pattern that helps them match and spot the code better and understand it better.

1. **Before** - I usually just use an automatic formatter in an extension in the IDE to take care of how I format my code so I never realized its importance.

**After** - the advantages of having a layout that’s easy to scan, expressive and has a compact format benefits programmers in multiple ways that makes understating the code much easier and faster.