**CHAPTER 1: CLEAN CODE**

**SUMMARY**

In a program, one should consider a standard or format to write a good documentation. Programming on the other hand has rules that every programmer should take it seriously. But before we go through the deeper understanding of writing a clean and efficient code, let us define first what is a code, how useful it is, and how to write a well document of it. *Code* in the dictionary is defined as a language that is understood by the computer. It is expressed in requirements. *Program* is a set of instructions that is use for the execution of a certain code. Creating a system has two outputs: it is either it will work and function properly or otherwise.

**The Primal Conundrum**

Every programmer faces a conundrum of basic values. Primal conundrum is a normal experience of developers when creating a project due to a particular situation. For instance, all developers with more than a few years of experience know that previous messes slow them down. And yet all developers feel the pressure to make messes in order to meet deadlines. But the only way to solve is to have a clean code.

How can we differentiate a clean code and a bad code? Below is the figure of where we can determine what is a clean code and a bad code.

|  |  |
| --- | --- |
| CLEAN CODE | BAD CODE |
| Readable and easy to understand. | Senseless code |
| Organized | Messy |
| Efficient | Complicated |
| Focused | Tries to do too much |

Those are just a few indications in naming the whole format of the code. Defining an art has complexities depending on the perception of the one looking at it. In terms of coding, distinguishing a clean code has several descriptions which relies to the perspective of the one who write the code. The following are the inventors of various mechanisms in the world of coding and their definition of a clean code.

**Bjarne Stroustrup** (inventor of C++ and the author of the C++ Programming Language)

Stroustrup states that when coding he likes his code to be elegant and efficient. *What does he mean by that? What is the connection of elegance in the field of programming?* He thinks that when writing a document or code, it must be pleasing to read. Moreover, clean code must be well-crafted and easy to understand. He also mentioned about efficiency. *What does he mean by that?*  A clean code must be efficient and well organized. It must be fixed on the process of paying attention to details.

**Grady Booch** (author of the Object-Oriented Analysis and Design with Applications)

Booch justifies that a clean code should read like a well-written prose. He uses the analogy of reading a good book. When reading a good book, it involves excitement and interest to details. He added that r code should be matter-of-fact as opposed to speculative. It should contain only what is necessary. Our readers should perceive us to have been decisive.

“**Big” Dave Thomas** (founder of OTI, godfather of the Eclipse strategy)

Dave asserts that a clean code makes it easy for other people to enhance it. He values small code and believes that it should be literate. In addition, code should be composed in a form of readability.

**Michael Feathers** (author of Working Effectively with Legacy Code)

Feathers stated that a clean code is a code that should be taken care of. It should be simple and organized.

**Ron Jeffries** (author of Extreme Programming Installed and Extreme Programming Adventures in C#)

Clean code must have no duplication, one thing, expressiveness, tiny abstractions. He also added that reduced duplication, high expressiveness, and early building of simple abstractions makes a code clean.

**Ward Cunningham** (inventor of Wiki, inventor of Fit, coinventor of eXtreme Programming. Motive force behind Design Patterns. Smalltalk and OO thought leader. The godfather of all those who care about code)

Cunningham stated that a clean code is easily identified by the output and the process while doing it. He added that a clean code is obvious, simple and compelling like it was made for the problem.

**CHAPTER 2: MEANINGFUL NAMES**

**SUMMARY**

A program requires a clear documentation that is organized and clean. To implement it, there are some certain considerations that is needed to comply. One of it is the “name”. Names are everywhere, even in different technologies. Naming convention is crucial in programming for it holds attributes that will help programmers understand the code.

**The Importance of Using Intention-Revealing Names**

 Names should reveal its intent. It should be detailed and specifies its purpose. If you choose a name that is abbreviated, not everyone will be able to understand it. For example:

Without the comment of “d” it cannot be identified, whereas we will consider this example:

With or without label, you can easily identify it for its directly states its description and purpose.

**Avoid Disinformation**

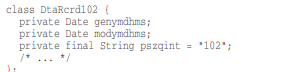
Developers must avoid leaving false clues that will hide the meaning of the code. In addition, words with entrenched meanings that vary from the intended meaning should be avoided. We must also take the spelling into consideration. How would you differentiate an information from disinformation? Using spelling in similar concepts is already an information, whereas using inconsistent spellings is disinformation.

**Make Meaningful Distinctions**

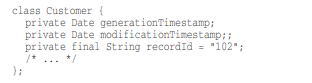
One of the problems programmers have encountered is they sometimes forgot the essence of naming when they try to write a code solely to satisfy a compiler or interpreter. Developers should also take naming variables in accordance to their meaning seriously and make it intentional for it is also crucial to make the program easy to understand. *Number series naming* *is* the opposite of intentional meaning. They are noninformative. Noise words on the other hand is also meaningless distinction. For example, you have class named **Product** if you have called **ProductInfo** or **ProductData,** you have made names different without making them mean anything different. Without proper distinction it will not be easier to understand and too complicated to trace. Take this as an example:

In my perspective, it is too hard to distinguish the appropriate method to call.

**Use Pronounceable Names**

 Humans are naturally good with words. In a literature, a piece must be readable by the audience. In programming, codes must also be readable and easy to understand by different programmers. Take this figure for example:

As a programmer, I cannot also understand what are the meaning of the variable names that was shown above, however if you compare this example to the previous figure:



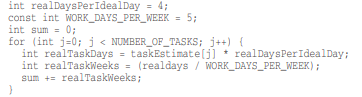
It is understandable and easy to know its purpose.

**Use Searchable Names**

 Using single names and numeric constant are not easy to locate. Take this figure for example.

Not everyone will understand it unless you are the one who created it. Numbers or single letter names are not advisable if you will use it for searching due to its various expressions where the value is used in different intent. Single letter also for it is the most common letter used in the English language and it is more likely to show up in every passage of the text.

In this example,



It is easier for me to understand for it states the complete description and purpose.

**Avoid Encodings**

Encoding type or scope information into names simply adds an extra burden of deciphering. In addition, encoded names are seldom pronounceable and prone to mis-type.

**Hungarian Notation**

Hungarian Notation and other types of encoding is simply an impediment. They make it harder to change the name or type of a variable, function, or class. They make it harder to read the code.

**Member Prefixes**

Prefix member variables are not really necessary anymore classes and functions should be small enough that you don’t need them. And you should be using an editing environment that highlights or colorizes members to make them distinct.

**Interfaces and Implementations**

Leaving interfaces simple can make users easy to understand. Too much information is not necessary when building a code. It must be precise and easy to distinguish the functionalities.

**Avoid Mental Mapping**

This problem generally arises from a choice to use neither problem domain terms nor solution domain terms. In addition, single-letter variable names are also a problem. Single-letter name is a poor choice; it’s just a place holder that the reader must mentally map to the actual concept. Professional uses power for good and write code that others can understand.

**Class Names**

* A class name should not be a verb.

**Method Names**

* Methods should have verb or verb phrase.

**Don’t be Cute**

Cuteness in code often appears in the form of colloquialisms or slang. Programmers should say what you mean. Mean what you say when writing a code.

**Pick One Word per Concept**

* Pick one word for one abstract concept and stick with it so that it will not lead into complications.

**Don’t Pun**

* Using the same term for two different ideas is essentially a pun. It will make the code complicated. Our goal, as authors, is to make our code as easy as possible to understand.

**Use Solution Domain Names**

* Choosing technical names for things according to the specific purpose is usually the most appropriate course.

**Add Meaningful Context**

* There is a need to place names in context for your reader by enclosing them in well-named classes, functions, or namespaces to make it easy to understand.

**Don’t Add Gratuitous Context**

* Shorter names are generally better than longer ones, so long as they are clear. Add no more context to a name than is necessary to make it readable.