**15.- TERAN ROBAYO CAMILA**

**Clean Code**

**A Handbook of Agile**

**Software Craftsmanship**

"In short, a programmer who creates clean code is an artist who can transform a blank canvas into an elegant code system." (Martin, 2009)

**Chapter 1 and 2**

Being a programmer and being a good programmer, no doubt both are on the same path but there is a big difference between the two, and we clearly notice it in their code. A code that is understandable to other programmers must be rigorous, formal, precise, detailed, that is, a clean code that meets the given requirements.

Like a clean code there is an incorrect code, where it is not possible to comply with an understandable structure, it does not comply with certain standards and the respective tests are not performed, thus generating a great loss of time and productivity. What causes this code is that other programmers cannot then modify it. Why at the end of your program does it end with a bad code? People are experts in looking for excuses and one of them are the requirements that change, the delivery time, etc. and we don't realize that the programmer must know how to act in all these situations and look for a solution, focus on his code and deliver in a correct and functional way; Without fear and without excuses, defend your code until the end and prove that it is a good code. What causes an incorrect code is more errors, if the problem is not solved from the beginning it will continue until all the code is finished.

A good programmer knows how to create a code suitable for changes and improvements (clean code), taking into account that the code must be detailed, understandable, organized and essential; We speak in this way of an elegant, efficient, simple and direct code, it fulfills what we expected. Having this type of code helps us so that over time the necessary changes can be made without any conflict or error and thus improve its version.

The first factor that we must take into account when creating a code is to write the names properly, this will allow other programmers to have no difficulty understanding the code. Their names must be clear, precise and specific so as not to cause any confusion and limit comments. When giving a name this should give us information about what we are doing, what is that variable, what function it is, what name I am looking for, enter other things and it is important to comply with a good structure when giving names, that is, use uppercase if it is a space or underscore, avoid encoders, use names with computer terms, the names must be different from each other to avoid confusion and thus help other programmers understand the context of the program and

be readable.

**Chapter 3 and 4**

Using functions in a code allows it to not be too long and makes it easy to verify that the code meets your requirements, and also allows the programmer to better understand the code. To be able to create really useful functions, we must take into account that the name must cover everything that the function will perform and that makes it easier for the programmer to understand the context of the code.

It is important not to mix details with essential concepts, it is better to raise from the beginning the direct concept that is going to be made and forget the details since the programmer just by reading the precise name of the function will understand. The descending rule explains that you should read from top to bottom to have a better understanding of the steps to follow that the code will perform, in other words, it is like descending on a ladder and going step by step. The extension of a function is very important since they cannot be extensive, the less lines they have, the better for the programmer, that is why a function must be concrete, to perform a single task. The correct name of a function improves its explanation and avoids confusion at the time of programming.

The use of comments may be unnecessary in our code, if the name of each variable and function is well described, adding comments is creating a bad code. If comments are used, they should be the minimum possible and these should be brief and precise.

If we need to return to a function to improve or modify, we use a reminder comment with the word TODO.

**Chapter 5 and 6**

A good format helps the programmer better understand the code, which is why the format must be clear. The name should be simple but clear, it reads from top to bottom and from left to right. The variables must be clear and placed at the top of each function so that the programmer can easily detect what the function will do and know where to look for them. The main function must be above the function invoked, a dependent function, and thus a clean and fluent code will be created, and goes from the upper to the lower levels. The function and the arguments are related to each other since the one is helped by the other. The structure of a code is very important, that is why we need to take into account the indentation, so that the code is more understandable. The methods of a class are indented one level to the right of the method declaration, thus generating an attractive view for the programmer. The most important thing is to have established rules within a group of programmers so that there are no conflicts and misunderstandings.

OBJECTS AND DATA STRUCTURE

Objects hide their abstraction data and show functions, while data structures show their data and lack functions. Under Demeter's law an object must hide its internal structure and an object must call another that is related. Avoiding hybrid structures, look at object and half data structure, generate confusion in the programmer since it is not known whether it is public or private.

The objects show behavior and hide data, this makes it easier for programmers to include new types of objects without changing the code. If the programmer needs to add data types, it uses objects, but if you want to add new compartments, add data types. It is important that the programmer knows what he wants for his code and use the best tool.

**Chapter 7**

ERROR HANDLING

To have a clean code and it is important to observe and analyze the errors that the code has. It is advisable to generate an exception when detecting an error. By validating each error the code allows the program to be more complex. One of the exceptions tools is try where the orders you want to give in the program are made and catch is used to capture the exception that is generated and show the user what is happening. To better understand the code it is important to comment on the exceptions, with the operation failed and the type of failure. Errors are classified by origin or by type. We must register a hero and make sure we can continue. A rule to follow is not to return null since if it does not comply with all the checks that the program must have, this may cause inconsistency. Another rule is never to pass null to another function as an execution error will continue to occur.

To avoid using an exception it is better to create a special class.

Para tener un codigo limpio y es importante observar y analizar los errores que tiene el codigo. Es recomendable generar una excepción al detectar un error. Al validar cada error el código permite que el programa sea mas complejo. Una de las herramientas de excepciones es try donde se realizan las ordenes que se quiere dar en el programa y catch sirve para capturar la excepción que se genera y mostrar al usuario que es lo que esta pasando. Para entender de mejor manera el código es importante comentar las excepciones, con l aoperacion fallida y el tipo de fallo. Los errores se clasifican por origen o por tipo. Debemos registrar un erroe y asegurarnos de poder continuar. Una norma a seguir es no devolver null ya que si no cumple con todos las comprobaciones que debe tener el programa este puede ocasionar inconsistencia. Otra norma es nunca pasar null a otra función ya que se seguirá produciendo un error de ejecución.

Para evitar utilizar una excepcion es mejor crear una clase especial.