**PAEZ BOLAÑOS FREDDY JAIR SUMMARY**

**Summary of Chapter 1: “Clean Code”**

In the first chapter this amazing book, tell us about how can be a great programmer, also It tells us if our code is correct or not. One of the most important issues is the total cost of a failure or disaster, since it tells us that when a disaster occurs the most optimal solution for it is to increase the workforce, the same that is one of the worst mistakes that can to commit because by increasing it, the new members will not understand the design or the objective of the program, which will increase many more errors and the disaster increases and productivity decreases until you get nothing. Many times at first glance our code looks of quality but this changes from one moment to another to an incorrect code, chapter one mentions that this occurs mainly because the requirements change what causes them to compromise the code itself, as well as various factors such as the delivery time that commits the programmer to finish the requested changes to the code in a certain time. One of the biggest problems that programmers face is that they do not know how to act if a disaster slows down their work, since those who do not have extensive experience the pressure plays against them and do not know how to act by making the delivery time lengthen On the other hand, professionals who have various experience, mention that this problem is not true since the disaster generated by a force majeure will cause the deadline to be stopped, and that the only way to move forward without pausing is to do from the beginning a clean code.

**Summary of Chapter 2: “Meaningful Names”**

In this chapter the central topic is about the name that should be given to the variables, functions, etc. since it is said that if we choose the well-centralized names, the next programmer who chooses to edit our code will make it much easier to understand and In many cases I would even appreciate it, that's why you should choose names according to what is being programmed. It is also mentioned that a programmer should not leave a variable in “poor” terms because it can cause a total loss, since the so-called false leads would be born. One point that I thought was very important was the spelling, it tells us that the correct spelling is information and that on the other hand the incorrect spelling is misinformation, by telling us this, we would be taking advantage of our time, since if we get the name of the variables are ordered alphabetically the user or programmer can search for an object by name without reading the comments that exist. The names given to the variables must be different despite the fact that a name is used and a series of numbers is added, it is not enough to do so since we would be creating a total misinformation, that is why the names must be different and at the same time they must have a different meaning.

**Summary of Chapter 3: “Functions”**

In this chapter the topic is the functions, it tells that in antiquity the programs were created based on other programs, subprograms and functions and that until now the only thing that has survived is the functions, something that is very important is that the functions they must be reduced, but when using this it is not proven or proven that having it so the study or practice is much more efficient. Over time, lines of code of 3000 lines and others of only 20 or 10 have been created, but based on trials and experiences it has been proven that it is much better to have a very small code. In this chapter a very attractive program called Java / Swing was used, which at the time of creating functions these had two three to four lines of code and all were very obvious, all the lines of length had an attractive order. This means that the main instructions such as: "if, else" must be at least one line in length, which will be the invocation of the function also implies that the nested function structures must not be extensive, and thus the function It will be much easier to read and understand. Many people want the descending rule to be applied to their code, which is that after advancing function by function, the level of abstraction does not increase what makes the code very easy to understand. but for programmers this rule is very difficult to implement, that is, to carry their functions or lines of code at the same level of abstraction, but it is also a very important point since they will reduce the length of the code and make the functions fulfill only one thing. In this chapter, another important issue is that of giving correct names, it tells us that we should not fear giving a long name to a function because the more long is the name that has been given more description, we must bear in mind that we must take time to choose a suitable name, because for that we must try different names and read the code with each name we have put, and we must do this until we find the name descriptive enough. if we talk about the arguments we have several: the monadic, dyadic and triadic in this it mentions that any of these can accept arguments that are variable, but also mentions that this could generate an error, and to finish this chapter the most important thing is the “ they must not be repeated ”because there can be four equal functions fulfilling the same objective and, when wanting to modify, it will generate a danger of create an error will quadruple in all the code.

**Summary of Chapter 4: “Comments”**

In this very important chapter, by the way, he talks about the good use that should be given to comments in the code, since a good comment can be very useful for the better understanding of a code be totally harmful to our code causing total misinformation. This chapter tell us that while we have an incorrect code we believe that the best solution is to comment on all sides, but it is not the best way to have a clean and well developed code without many comments and if it is the case to have a code incorrect, we must try to solve it but not fill it with comments everywhere. Many times it takes us longer to explain what we want to do with the code when we can simply explain it with a well-posed function. A big piece of advice that the book gives us is that we always the one that never writes it. In some cases it is mentioned that the comments can also be legal, this means that in those comments there will be the copyright of the code, the same ones that will appear at the beginning of the code. Another of the issues that I stress are important are the comments "EVERYTHING" these are those that work as a reminder to the programmer, as to what they have to do, or if it is the case to solve an error, as a last interesting topic for me were the incorrect comments and the redundancy that is generated by having bad comments, if we talk about the bad comments it is about the comments that are not necessary to be in the code, this would be more or less as if the Programmer will talk to himself. and if we say about redundancy, this occurs when an unnecessary explanation is given in our code when at first glance with just a glance we can understand the code in many cases it takes longer to read the comment than to understand the code itself. One tip not to have a fully commented code is the worst thing that can be done, the best and most optimal is to avoid the use of comments in excess, as well as avoid comments with abundant information, which will make the code increase in length, generating a disagreement with the naked eye.

**Summary of Chapter 5: “Format”**

We have to worry about the format of our code we must choose a series of simple rules that control the format of the same pair to the rules for apply them correctly, in the case of working in a group you must specify to all members of the group that must work in based on the established rules or preferably work with a rules automatizer.

The code must be clear since the format is something very important so as not to take it into account and this is based on communication, which is the main thing in a professional developer, it is often said that the main concern of the programmer is that his program to execute but it is not like that because it is leaving the legibility of the code aside, at the moment of realizing this it affects the code compromised with all the changes that are made. in the vertical size it is specified that an average vertical code size is 200 to 500 lines but after all a small code will be much better understandable than a large length code. A code must be like a newspaper, that is to say, it must have a clear but simple name and while it is going to be modified, all the functions that must be performed are presented, the details must go while it is advancing in the programming so that in the end Let's have only the solution. a newspaper often has several sizes but if we saw a very large newspaper it would make us not read it is the same thing that would happen if we would have a very long code we would not be able to understand it and leave it aside.

We need to keep the concepts of our code in relation because being the case of having on all sides we will spend time and mind trying to understand without having any, that is why we should also avoid using protected variables. the variables must be declared more approximate to their use, as the functions are very brief they must appear at the top of each function. When talking about functions, they must be on top of each other, that is, vertically, as we have already mentioned, in order to achieve our program, the function that invokes other obligations should always be placed above the others in order to improve the readability of the code.

if we want to define our code with a horizontal format, a simple view will seem to have a totally clean code but no, when doing this type of code we are losing the good view of the code since having all the left-to-right run will cause us to not understand what is the intention the code wants to do.

**Summary of Chapter 6: “Data Objects and Structure”**

Objects hide their data through abstraction but show the functions that operate in that data. The classes are complementary but the nature of the same at first glance are totally opposite, all behavior will always be found in only one of the classes we have written. Demeter's law tells us that a module should not recognize the data of the objects that are being manipulated. As we can see in the previous section, the objects hide their data and operations or it is concluded that an object must not show its internal structure, and this must be done through processes since if we do not, it will be shown immediately All its structure. The hybrid structures tell us that the code must have the object half and the other half of the data structure since there is the possibility that there is a mutation that makes the variables that are private make them public, these hybrids are the ones that complicate the inclusion of new functions and at the same time of new data structures. the use of this should be avoided if the authors or programmers are unaware of it. Active records are a special OTD record (indicator), as is the navigation of find and save public variables, to make effective use of that the programmer must have an active record of a data structure and also create the independent objects.