GOOD CODING VS BAD CODING

Bad code is like an untidy room, you spend hours in finding things and when you try to add something, you are just adding mess. Good code is like a tidy room, you immediately find things, and you can easily change them.

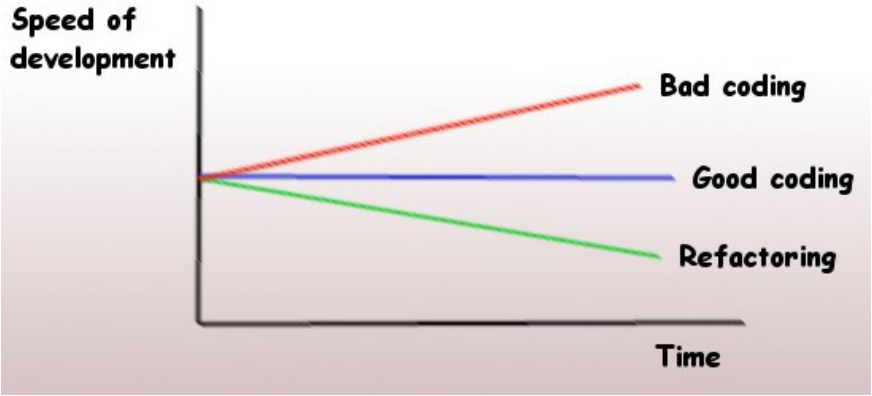
One day I visited an accountant, amazed for the tideness in the office. He had hundreds of customers and hundreds of documents for each customer, but as I requested one of the documents, the secretary finded it in thirty seconds. I asked her how she could be so fast and she told me they were very well organized. The same is true for coding: if you write good code, you can easily apply new changes in a few minutes, without spending too much time in understanding the code, in locating the lines of code to modify, or worrying about side effects.

A disorganized accountant can survive with one customer, maybe he can survive with ten, and you'd probably still get your document in two days... but he can't survive with one hundred. The same is valid for coding: you might have bad code and still be able to work, if you just have to maintain a small piece of code; you might have an hard time, if you have a slightly larger one; but you can't survive, if you have a huge software. One day will come that you have to rewrite it entirely.

The code quality is related to the speed of development, and a good developer is as fast as a well organized accountant, which can find and update documents very quickly. A fast developer that writes bad code, hovewer, is like a vagrant

with a shiny suite: he will look great to the management, but he will pretty soon slow down the entire project and will hazard the business.

Remember, the speed of development is not just the time you spend to develop a certain functionality, but the time others will spend in extending your code. The speed of development can increase over time, as you indulge in bad practices, but it can improve with refactoring.



Whenever you extend your codebase with new features, you add complexity and the time of development for each new feature can only go up, or remain the same, in the best case. This also happens when the accountant get new customers and new documents, or when you add something new to your room. You are adding more and more chaos. The solution is to reduce the complexity by making the code more simple and easy to maintain and understand. That's achieved by reorganizing things in a better way, to reduce complexity again. You cannot foresee how your software will change in the future, and the accountant cannot foresee which new laws he will have to comply with. The only way to cope with the increased complexity is reorganizing and simplifying things. In software development this is called refactoring. **Every time you refactor your code, you are reducing time and**

costs of future developments, and you are reducing complexity in the development process.

Now, if you write good code, without refactoring, complexity can still slowly go up. In the best case scenario, it will remain stable, but it will never go down. However, if you indulge in writing bad code, complexity will skyrocket and you will be soon in trouble.

When developers are only driven by the speed, they will be slower and slower. When they are driven by the quality, they will be faster and faster.

Software is not something printed in hard rock, that will never change, software is dynamic and must be maintainable. Everything has been said about good practices and good coding can be reduced to one single factor: maintainability.

When the code is maintainable, it is also good code. If it is good code, it is easy to change, easy to undersand and keeps the costs low.

It also drastically reduces bugs, for the same reason an organized accountant will commit less errors. A clear design and a self-explanatory code are not error-prone, they tend to make errors so evident, that they look like a lighthouse in the middle of the night.

Sometimes the code can be so nebulous and the developers so thoughtless, that they spend most of the time fixing bugs. They keep fixing problems introduced the day before.

Here is a fantastic indicator: **if you notice that you are spending most of your time fixing bugs, stop one second and think about the quality of your code**. Probably you need to think to some serious refactoring, because that's one of the reasons your development will slow down and your project will fail.

How do you write good code then? Simple, all you need to do is writing maintainable code: you need to be able to find, understand and change things easily. Be simple, don't add complexity, and refactor your code.