11 basic principles that every programmer should follow

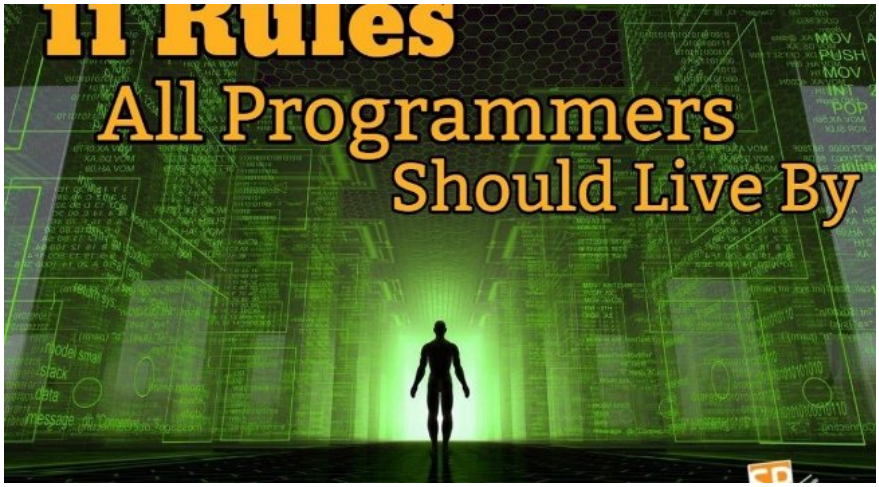
Welcome to read 11 basic principles that every programmer should follow in this article!

1. 13 famous inspirational lessons about life from Steve Jobs
2. 10 qualities NEED to help you succeed
3. 50 soft skills needed to be happy and successful for life (Part 1)

I am a person who tends to live by the principles. Still the same, in fact, they are mostly principles that I set myself up for - but that's still the rule, at least my own.

I found that creating the principles for myself helped me to live and work better because before I decided on something I **spent time thinking about what I needed to do instead of coming anywhere or coming. that** .

For example, should I go to the gym this morning? Yes, my rule says that on Wednesday I have to go to the gym and today is Wednesday, so I'll go to the gym - that's for sure.

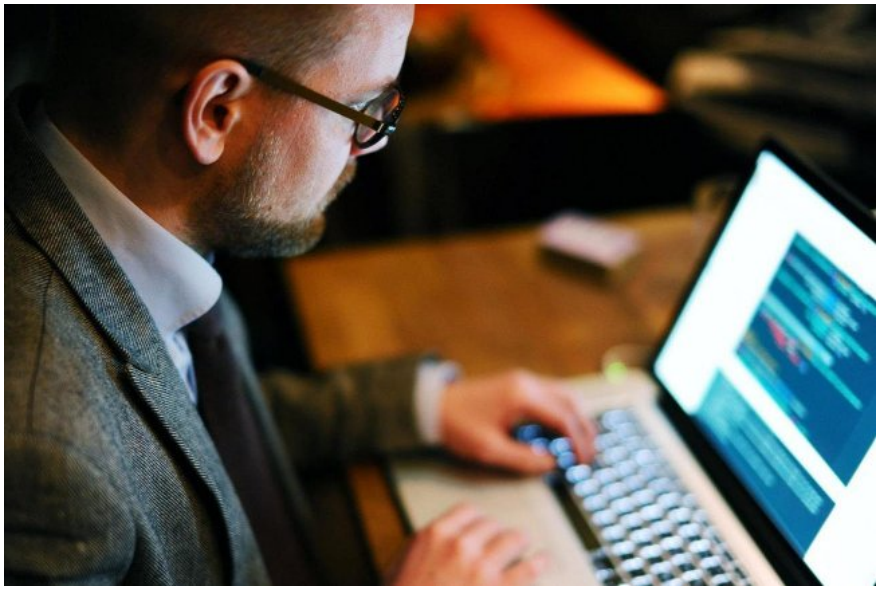


This week, I am thinking about some of the principles set for myself, I think it might be a good idea to create a set of rules that all software developers should apply in work and life.

I admit that most of these programming principles are guidelines and I will list them below:

1. Technology is the way you find solutions, not solutions

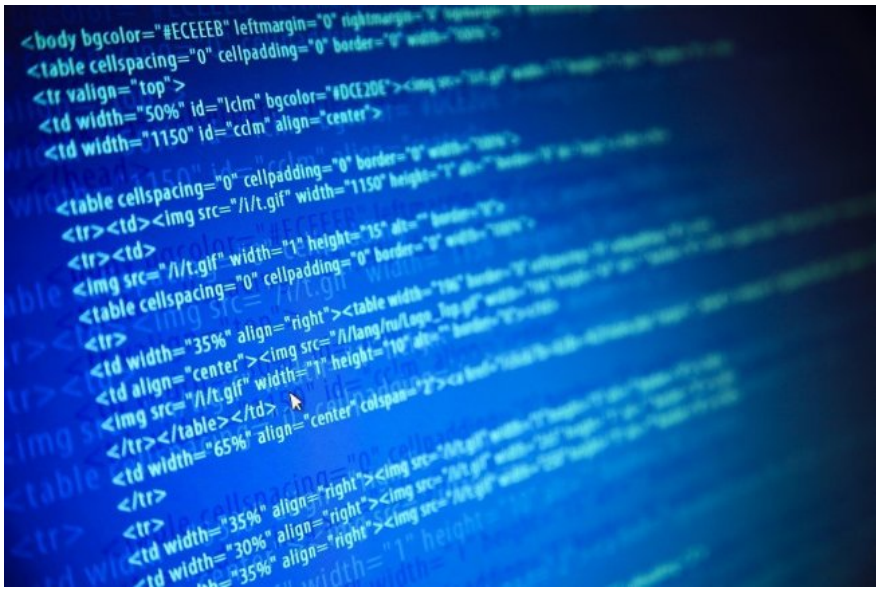
We can actually have the hottest **JavaScript frameworks** like Angular - IoC containers, programming languages ??or even operating systems, but all of these are not really solutions to problems; because we are trying to solve it as a programmer. Instead, they are simply tools to help us solve the problems we are facing.



We have to be very careful not to be too extreme about a particular technology that we like or it is popular now, otherwise we risk thinking about all the issues like one. nails, just because we are holding a shiny hammer that we just learned about it.

2. Intelligence is the enemy of clarity

Have you ever encountered this situation: There is a need to fix, or a huge bug. You think of a very smart, brief and self-tapping solution that praises me smartly. But then one month later, reviewing the code (*of myself*) does not know how my ' *smart* ' code runs and how to fix it?



When writing code, we should try to write so that **it is clear and understandable** . The code clearly conveys a purpose that is far more valuable than obscure code - no matter how smart it is.

Of course this is not always true, but in general, **intelligence is the enemy of clarity** . Sometimes we write code that looks ' *smart* ', but that code is not particularly clear.

It is important that we remember this rule whenever we think we are doing something especially smart.

Sometimes we write code that is both smart and clear, but often it rarely happens.We are programmers, **the quality of code is measured by clarity and maintainability; not the less the code is, the better it is when you go to school or study** .

If you are interested in writing clear code, I recommend reading the book " *The Clean Code: A Code of Conduct for Professional Programmers* " (*roughly translated: Coder Code for Professional Developers).*) by Robert C. Martin.

See also: 13 Best free eBooks for Web Designer

3. Don't write excess code, write exactly what you need to write

When I read it, this seems to be a bit contradictory to number 2? But isn't all the programmer's job after writing the code?

In fact, the answer to this question is "yes" and "no".

Programming can involve writing code but we still need to try to write as little code as possible to solve the problem we are having.

Of course this does not mean that we should make our code as short as possible and name all our variables using the letters of the alphabet, but try to **write only the code. when really needed** to perform the required function.

Usually you add all kinds of features to your code or make our code ' *strong* ' and ' *flexible* ' so that it can handle all different situations. But most of us are wrong to try to guess useful features or anticipate problems that we think might be encountered in the future.

The code we write to anticipate future situations may not add any extra value. In contrast, many code also produce more bugs and maintenance is more difficult.

Good software engineers don't write code unless it's absolutely necessary. Great software engineers often delete as much code as possible.

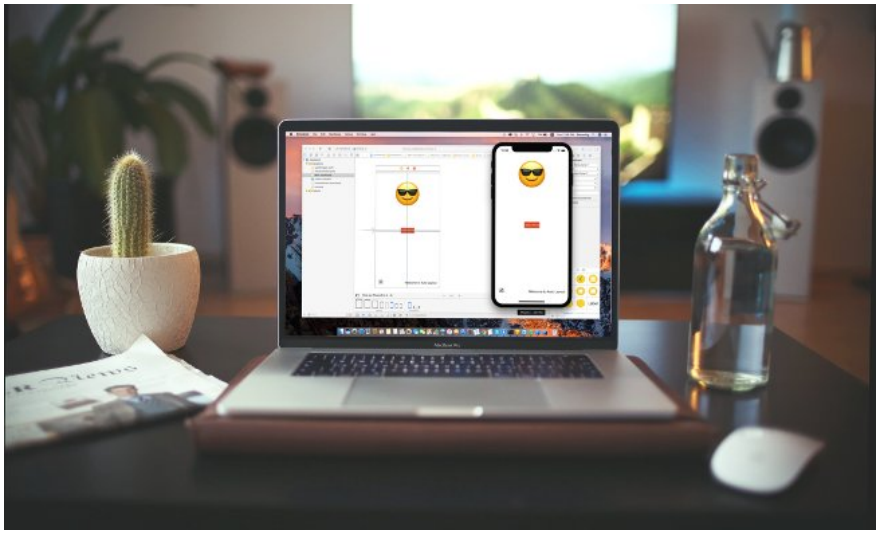
4. Abuse of Comment in most code is not good

I'm not a big fan of commenting in code. I also agreed with **Bob Martin** , when he said:

' *Every time you write a comment in code, you should frown and feel the failure of your expression* .'

- Clean Code: A Handbook of Agile Software Craftmanship

This does not mean that you should never write comments in code, but for most cases they can be avoided and instead you can focus on improving the naming of your variables and functions. to express meaning better.



Comments should only be actually written when you cannot explicitly convey the intention of a variable or method through their name. Comments that serve to code parts are not easy to express themselves.

For example, a comment might tell you about some strange activity that takes place in code but not an error, but the intention of the code writer.

In general, comments are not only harmful because in many cases they are necessary, but sometimes they do not properly describe what the code that comes with it is being done.

Comments that are not updated with the accompanying code, these comments become really dangerous, because they will probably drive you in a completely wrong direction.

Do you regularly check all comments against the code to make sure that the code is actually doing what the commenters say? If so, what is the purpose of having those comments? If not, how can you believe those comments are true?

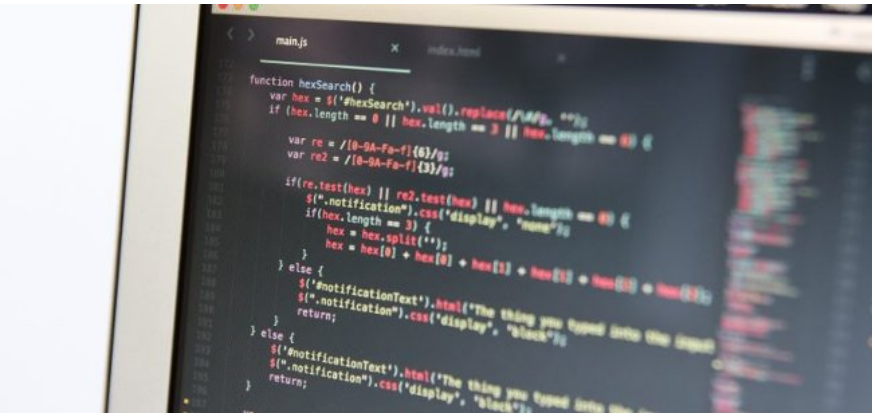
That's a double-edged sword, so it's best to avoid using comments as much as possible.

If you disagree with my point of view, leave your ' *comment'* in the comment section below the article, but I - the author, will not change my position.

See also: How to write commands, indent and annotate in Python

5. Always know what your code should do before you start writing it

This situation is quite familiar, probably not many readers have encountered. Sometimes, we plug in the code input before clarifying the problem. Indeed there are many times when we have to code to know what to solve and how to code. However, the advice here is: **Determine the purpose of the code to be done before the code** .



This looks obvious, but it is not so.

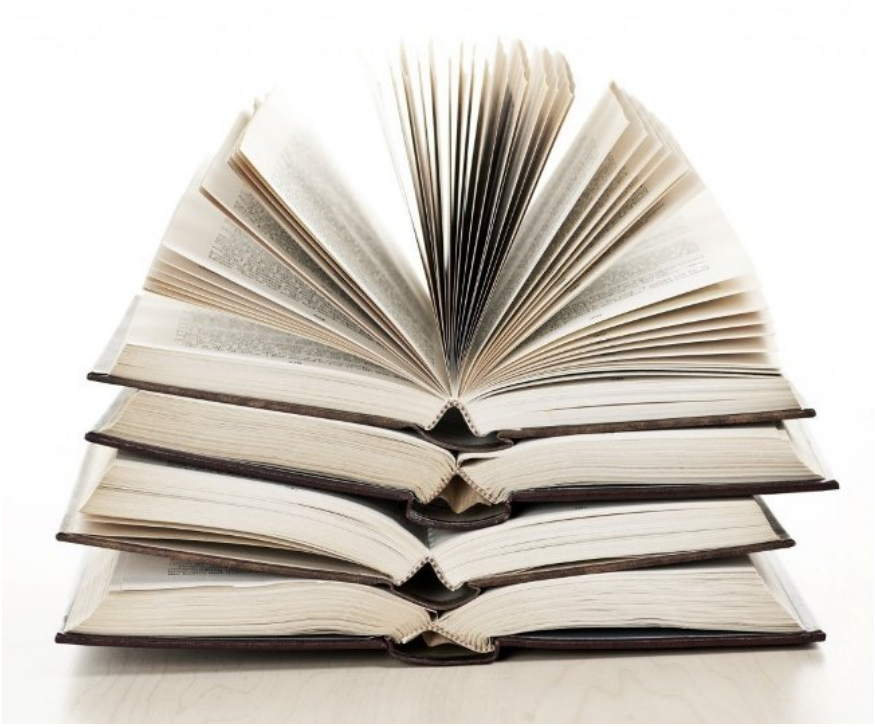
How many times have you sat down to write code without really understanding what code you're writing really has to do?

I have to admit that I have fallen into this situation many times, so this is a principle that I need to re-read often.

Test Driven Development (*TDD*) can be useful in this case, because you have to know what the code will do before you write it, but it still doesn't prevent you from creating things. false. Therefore, it is important for you to fully understand the functionality and functionality requirements you are building before you start working.

6. Check the code before handing it over

Test your code before throwing it to the tester! Of course, testing is the tester's responsibility but a programmer ' *ethical* ', test the basic case yourself, fix silly mistakes, unnecessary to avoid wasting everyone's time.



Don't throw your code to the Tester team as soon as it's done, because people will send you a bunch of bugs and you'll lose your time and everyone in creating bug reports and resolutions. necessary.

Instead, take a few minutes to test your tests, before you think you've finished the job.

Surely you will not catch all the errors before moving your work to the Tester team, but at least you catch some silly and embarrassing mistakes repeating this time over and over. Other times.**Ensuring product quality is everyone's responsibility.**

7. Learn new knowledge every day

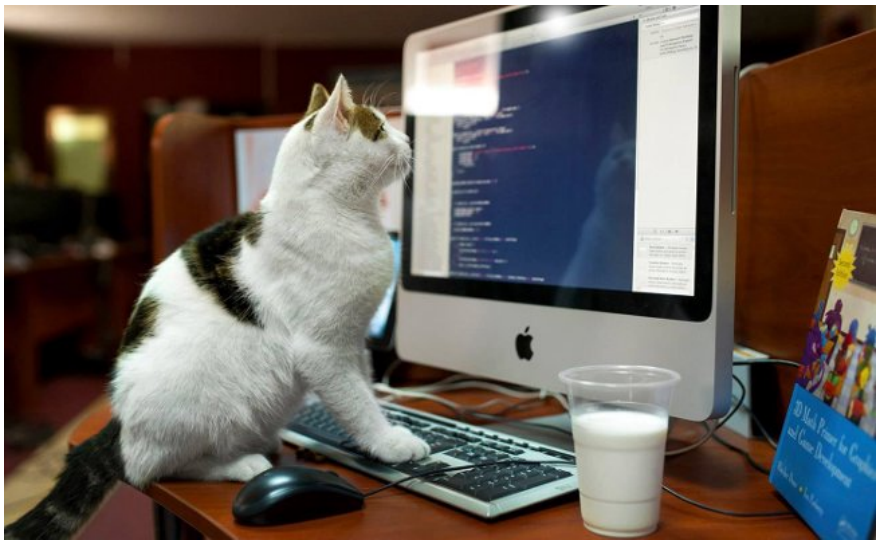
I'm sure you'll forget some knowledge while technology is changing every day. It won't take too long to learn new knowledge every day. Try to take 15 minutes or longer to read a book, I read a lot of books last year, averaging 45 minutes a day on average.

The small technological advances that you make every day over time will give you a lot of great opportunities in the future. However, you must start now if you want to reap the rewards.

Today's technology is changing rapidly, if you **constantly improve your skills and learn new things** , you will be left behind very quickly.

See also: 8 useful tips on work for young people

8. Writing code is very interesting



Yes. You may not have decided on this industry just because it has a good salary. Honestly, career as a programmer is a job with a fairly good salary, the job is also easy to find compared to other industries.

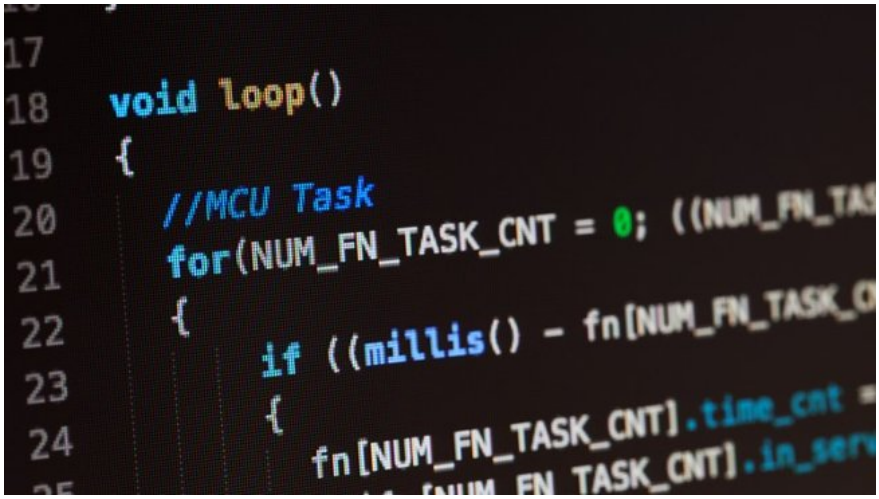
Most likely you become a software developer, because you like to write code. So don't forget that you are doing what you love. Writing code brings a lot of fun. I wish I had plenty of time to sit and write code.

I am often too busy to maintain my business but less time spent writing code every day, which is one of the reasons why I remember very clearly the endless joys that work write the code to bring.

Perhaps you have forgotten how happy it is to write code? Perhaps this is the time to remember the joys you had, by starting a *side project* or just changing your mind and realizing that you can write better code, even even paid for that job.

Why do I study code instead of pursuing a career in finance?

9. Accept that not everything you know



Don't try to learn everything, don't try to show yourself everything . You can say ' *I don't know* ', can also ask others when there are unknown issues.

The interesting thing is that the more you learn, the more you will find that there are still many things that I don't know. It is important to recognize this because you may be trying to know everything.

Also OK if you don't get all the answers. Ask for help or ask others when you don't understand something.

In many cases, you can learn about what you need to know at a time when you need to know about it - believe me, I've done it many times already.

My point is: **Don't try to learn everything, it's an impossible task** . Instead, focus on learning what you need to know and building skills that help you learn everything quickly.

10. The best solution depends on the situation

It is not always recommended to apply 3 classes, apply IoC, apply Test Driven Development. Although they are called ' *best practices* ', we must apply them according to circumstances, not just blindly applying them.

Is **Test-Driven Development** (*TDD*) the best method to write code? Should we apply *pair programming* ? Do you feel inferior if you don't use IoC containers?

The answer to all these questions is ' *optional* ', depending on the situation encountered.

People will try to push " *best practice* " down your throat and tell you that they always apply them - that you should also follow them - but, that is simply not true.

I have followed a lot of the best solutions when writing code, but besides, I also set out the conditions to know when to apply and when not.**The principle is forever, and the best solutions will depend on the specific situation** .

See also: 25 extremely useful websites and applications will definitely make you smarter

11. Always towards simplicity

The best way to solve a problem is often the simplest way (*Don't do the 'clever' solution mentioned above*). However, this ' *simple* ' way is sometimes very labor intensive to find out. Before giving a complex solution to a problem, ask yourself: ' *Is there a simpler way to solve it?* .

All problems can be broken down and solved. The best solutions are usually the simplest ones.**But simple does not come easily** . You have to work hard to make things simple.

The purpose of this blog is to make complex issues in software development and life more simple.

This is obviously not an easy task. Any " *idiot* " can create a complex solution to a problem. More effort is required and determined to refine the solution to make it simpler. **Spend time, make great efforts and strive for simplicity.**

If you have to add a 12th rule, it is: **Make yourself justice** . Great programmers do not always lead others and demand the best pay. Why? They don't " *market* " themselves.

What principles are you living with? These are my principles, what are your principles?

Which principles do you personally live with? Do you think it's important to remember every day and practice it as a habit?

Please share your principles in the comment section below!

Author: John Sonmez

See also: 13 skills needed to become Frontend Developer

Having fun!