Developers Best Practices

1) What's practice?

When I'm saying "Practice", what does it mean? I would say:

- Practice is **a habit.**
- Practice is **a routine.**
- Practice **does not need to remember.**
- Practice **comes by practicing.**
- Practice **needs dedication and commitment.**

There are thousands of examples which you think about practice. I can list few for your understanding.

Shooting, Driving, Writing

Any of the above listed skills comes from practice. When initially you start driving, you need to remember each step and you think twice before taking any action, but once you "have good practice" of driving, then you do not need to remember any step. It becomes your habit and routine, for example, your feet goes automatically at brake if you see a red light but definitely it comes from practicing a lot and needs a lot of dedication and commitment.

One of the most important attributes of practice is that it **forces you not to divert** from what you used to do.

There could be a driver but would you assume him an efficient driver if he is driving at a speed of 20 miles per hours and meeting with accidents so frequently and bringing lots of scratches in the car on a daily basis?

Software development is also not different than other skills like shooting, writing or driving. To become a **successful** software developer you need lot of practice, dedication and commitment.

Through this small article, I'm going to tell you few major best software developer's practices, which you may find useful. So let's start....

Best Practice 1- Keep Reading Existing Software Source Code

Let me ask you few basic questions before we start with one of the most important best practices required for a software developer.

- Do you read movie magazines?
- Do you read newspapers?
- Do you read roadside advertisements?
- Do you read junk written here and there?
- Do you just read....?

Definitely your answer will be positive but if I ask you one more question in the series:

Do you read Software Source Code?

Only few software developers will have positive answer because reading and understanding an existing software source code is the most boring task. If you are one of them who feels reading software source code is a boring task, then you are missing one of the most important best practices, which a software developer should have in his/her life.

If you want to become a novelist, can you just start writing novels? I would say 100% no!!, you definitely need to read hundreds of novels before you start writing **GOOD** novels. If you want to become a movie script writer, can you start writing good movie scripts until you have gone through various good movie scripts?, again my answer would be no!!

So, if you want to write a good software code, then how it will be possible for you to write a good source code without reading tons of source codes? Even if you will write something, then how would you and know which the best is?

Reading source code written by others gives you opportunity to criticize the mistakes done in writing that code. You will be able to identify the mistakes other software developers have done in their source code which you should not repeat.

There are many attributes of software codes (indentation, comments, history header, function structure, etc.), which you will learn by reading existing code, specially, a code written by well-experienced software developers. Spend some time in reading others' source code and I'm sure you would be able to write **BEAUTIFUL** source code in few days or few weeks and you will be able to fix the mistakes, which you were doing so far in writing the source code.

One thing to experiment, just go in the past and check the code you had written few years ago, you will definitely laugh....because you are always improving by doing practice.

Best Practice 2 - Complete your documents before next step

I had passed out my masters in Computer & Application and I was so passionate to write source code even without completely understanding and documenting the requirements. Design document and test cases documentation were nowhere in the software development life cycle ....there was direct jump to the coding.

At later stages I found myself in big trouble and soon I realized **Documentation is the Key** to become successful software developer, tester or architect.

Before you start developing small or big software, you should have answer for the following questions:

- Where is the Requirements Specification?
- Where is the Impact Analysis Document?
- Where is the Design Document?
- Have you documented all the assumptions, limitations properly?
- Have you done review of all the documents?
- Did you get sign off on all the documents from all the stakeholders?

Once you have positive answers for all the above questions, you are safe and ready to proceed for the coding. Many organizations would have strict rules to be followed, but others would not have. Best practice is to complete all the required documentation and take appropriate approvals before proceeding for the software coding.

What you learn today, prepares you for tomorrow!

So, again it is one of the best practices to have documentation as much as possible. Few important documents, which will prepare you for future are:

- Design Approaches
- Tips and Tricks

- Special functions, commands and instructions
- Lessons learnt
- Peculiar situations
- Debugging methods
- Best Practices
- Anything which can help you in future

Keeping documents electronically does not cost you. So let's start maintaining required documentation.

Best Practice 3 - Follow the defined standards, don't create it

Most of the standard software organizations maintain their coding standards. These standards would have been set up by well-experienced software developers after spending years with software development. This is equivalent to following footsteps of great people left behind them.

If your organization does not have any standard, then I would suggest to search on internet for coding standards off different programming languages and you will find many. A coding standard would fix the rules about various important attributes of the code, few are listed below:

- File Naming convention
- Function & Module Naming convention
- Variable Naming convention
- History, Indentation, Comments
- Readability guidelines
- List of do's and don'ts

But once defined, start following the defined standard instead of creating or changing them every day. I would definitely say:

Source code is your BABY!

So keep it clean, consistent and beautiful. When I say beautiful, it really means beautiful. If your code looks beautiful, then it would be easy for others to read and understand it. If you will keep changing coding rules every day, then after few days you, yourself would not be able to read and understand the code written by you.

Best Practice 4 - Code should be written to be reviewed

While writing your software code, keep in mind that someone is going to review your code and you will have to face criticism about one or more of the following points but not limited to:

- Bad coding
- Not following standard
- Not keeping performance in mind
- History, Indentation, Comments are not appropriate.
- Readability is poor
- Open files are not closed
- Allocated memory has not been released
- Too many global variables.
- Too much hard coding.
- Poor error handling.
- No modularity.
- Repeated code.

Keep all the above-mentioned points in your mind while coding and stop them before they jump in your source code. Once you are done with your coding, go for a self-review atleast once. I'm sure, a self-review would help you in removing 90% problems yourself.

Once you are completely done with your coding and self review, request your peer for a code review. I would strongly recommend to accept review comments happily and should be thankful to your code reviewers about the comments. Same time, it is never good to criticize any source code written by someone else. If you never did it, try it once and check the coder's expression.

Accept criticism but don't criticize

Poorly written source code teaches you to write good source code provided you take it positively and learn a lesson from it.

Your target should be to stop the bugs at first place and create a BUG-FREE code. Think like a tester, so that you should have a challenge for the testers.

Best Practice 5 - Testing to be followed like a religion

Testing is mandatory after every small or big change no matter how tight schedule you have or you just changed a small comment inside the code, you have testing due for the changed code.

There is nothing like trust while developing software, no matter how expert or how senior you are in writing source code, you would have to perform testing for each and every change you did in the code.

- Tight schedule, no compromise.
- Changed just a comment, still you have to test it.
- Changed just a variable name, testing has to be done.
- If you feel lazy...it's too dangerous.
- If you don't want to follow it? You will be in trouble!

Celebrate every bug you find

Yes, you should not feel unhappy if you or another tester finds a bug in your software source code. Following are the enough reasons to celebrate this important discovery:

- Bugs are your enemies, so you have killed one.
- Now your software is having one bug less.
- Mistakes are good as long as they are not repeating.
- What you learn today, prepares you for tomorrow

Same time, do not criticize any developer in case any bug arises in his/her code because so far at least I do not know any programmer, who can write bug-free source code in the world, second this is one of the reasons we have a separate phase in SDLC (Software Development Life Cycle) which we call post production support (or support & maintenance).

Best Practice 6 - Keep your Code and Documents Safe

A smart developer keeps habit of taking daily backup of the produced artifacts, otherwise machine crash can crash you as well. You should keep your artifacts at your local machine as well as another secure machine, so that in case of machine crash, you can continue with the saved copy of the source code or documents.

If you have the habit of taking daily backup then in worst scenario you may lose at most one-day effort, but if you take weekly or monthly backup, then there is a risk of losing whole-week or whole-month effort, and you will face biggest disappointment you ever had.

Multiple copies create confusion

This is true that having backup is one of the most important best practices, but it should be maintained in well managed way as you can use tags like name, date and time of the backup, version, etc. If you have multiple copies of the same source code or document, then it will create confusion and it would be difficult to identify latest code or document.

It is strongly recommended to use proper source code version control system. There are many source code version control software applications available for free (like SCCS, CVS, Subversion etc.) which you can use to store different versions of the software. But while using a source code control system, follow the rules below:

- Always take source code from the version control system.
- Always assign a new version to every change.
- Always put source code back into control system.

Password sharing is strictly prohibited

- Love, affection, friendship and relationship are on top of everything, but never embrace anybody asking for password.
- If you are sticking to first point, then why you would share your password with anyone if you are not asking from anybody.
- Keep changing it on a frequent basis and it's good if you have some logic to drive your passwords, otherwise during your long vacation, you will forget them.

Best Practice 7 - Keep your Tools & Techniques Handy

I remember an instance when I wanted to find out **debug** keyword in all the C++ files available in various directories and sub-directories, it took me 30 minutes to find the command, but finally, I kept a note of the command, and whenever I'm in need, I use it without wasting a second.

```
$find . -name \*.cpp -exec grep -q "debug" '{}' \; -print
```

So, I made it one of the best practices to keep such commands and tools handy so that they can be used anytime without doing any R&D and to save valuable time. Better to maintain a text file having all such frequently used commands and create its link at desktop.

Few Essential Tools

It depends on what type of programming, coding you are doing but following are few of the essential tools, which should be readily available with a software developer:

- A good text editor to write and edit the program.
- A nice debugger to debug the program.
- A memory detector in case you are using dynamic memory allocation.
- Putty to connect to a remote machine.
- WinSCP or FileZilla to ftp files on a remote machine.
- IDE (Integrated Development Environment) for rapid development.

Always keep adding new tools & techniques in your box

Make sure you keep applying latest patches of your tools and utilities and same time I will suggest to clean unwanted software from your computer as they unnecessarily make your computer slow and you never know if any one of them is having a security hole, which can expose your computer to the outside world.

Best Practice 8 -Leave the ego behind, Be eager to learn

We always learn from books and nowadays from internet. But IT is such a field, where we learn a lot from our colleagues. They are our best references, but there are software developers, who either feel shy in asking their doubts or are not thankful to others, so ultimately when they ask next time, they get zero answer.

IT is vast and nobody can have complete knowledge on any subject. Every day, we come across different problems. So Ask...Don't feel shy if you don't know X.

I'm not suggesting you to bother someone unreasonably and asking for spoon feeding to learn anything. NO, be polite, thankful, directly come to the point, understand and support others.

New technologies are coming everyday

If you want to sustain in the market, then you would have to keep yourself updated with latest IT tools, and technologies. Following are the few sources:

- Technical Forums over the internet.
- Technical magazines on various IT subjects.
- Technical Bulletin Boards
- Conferences, Trainings and Workshops
- Latest versions of old tools and packages, languages, etc.