PROGRAMMER'S MOTIVATION FOR BEGINNERS

Real Learning Stories & Tips

By Rajaraman Raghuraman

Programmer's Motivation for Beginners

About the Author

Rajaraman Raghuraman has 8+ years of experience in the Information Technology industry focusing on Product Development, R&D, Test Data Management, Functional and Automation Testing. He has architected a TDM product from scratch and currently leads the TDM Product Development team in an IT MNC. He is passionate about Agile Methodologies and is a huge fan of Agile Development and Agile Testing and all aspects surrounding Programming, Software Development & Testing, Project Management & Products. He loves to share his knowledge to the community. He also owns a blog (http://agiledevtest.blogspot.in) and shares his thoughts about development, testing, products and startups in it.

INTRODUCTION

They taught me programming concepts, they gave me the lab for exercises. But they didn’t teach me any stuff what it will be like in the real time to do programming and what are their numerous challenges. And more importantly no one taught how to see code like an art. Things that I missed to learn in college, things that no one taught me during my early career days, I thought I would share some of those things to you so that it will benefit you.

WARNING: This E-book is intended for beginner programmers who are either starting to learn programming or are starting their programming careers. This is a compilation of some of my blog posts , some simple and practical advice and my learning experiences throughout. This E-book doesn’t contain specifics of any one programming language, its all generic. So its applicable to all programmers be it Ruby, C#, C, C++, JAVA, PHP, etc.

CHAPTER 1 – WHAT I LEARNT FROM MY FIRST REAL WORKING CODE

That was my first reasonable project in programming. I had written a parser for 2-pass macro assembler using C. It was around 2500 lines of code, I was happy that my code worked in my machine. Better yet, my code worked during the demo as well. That's a miracle. So I had a great milestone to brag about – THE WORKING CODE. But after I finished the project, I sat after a few weeks to just review the beautiful art that I had created (hehe, I am talking about the code that I had written). I was hardly able to understand anything in my own code !!!!!!!! :) That's when I learnt some important lessons in programming.

Modules please

My working code contained 2500 lines of C code in the main function. Wow… What an achievement, I thought. But when I reviewed it, I realized that things should not have been this way!!

Lesson Learnt: I should have split the code into nice modules, that would have saved a lot of time during maintenance.

Comments please

I didn’t put comments wherever necessary, so I was caught trying to figure out what each and every line or piece of code did. It was difficult as hell and I wasted a lot of time literally trying to find out what the code did. Remember, this was just few weeks after I wrote the code.

Lesson Learnt: I should have added enough comments for even myself to understand the code in the future

Segregation please

There were not enough segregation between variable declaration, assignment and re-initialization. That caused a lot of confusion when trying to read it through.

Lesson Learnt: I should have segregated the code into blocks of reasonable size and functionality.

Naming conventions please

I was great at naming conventions, I named variables as i1, i2, i3, i4, ii1, ii2, etc.. I thought I understood the code when I wrote it. But when I tried to review it, that's when I realized that I did a horrible mistake by naming the variables in a clumsy manner.

Lesson Learnt: I should have followed proper naming conventions for variables (No worries about functions, as I didn't even bother to write them in the first place)

The code is not just about me

When I was reviewing my own code, I thought "What if someone else would have to maintain this piece of GREAT working code?". They would have run away seeing the code in the first place.

Lesson Learnt: I should have written code that even someone else should be able to understand.

Both inside and outside of a software are equally important

My code worked great. I was able to show a demo. But what if someone had asked to review the code that I wrote. Luckily there was no marks for the way I wrote the code, so I escaped !!:) That's when I realized that both the presentation and internals of the software are equally important. Presentation is for the customers and internals is for the programmers.

Lesson Learnt: I should have bothered about writing code in a good way and focus on the internals also !!

Your code will bite you, if you don't take care of it

Sure it bit me many times when I reviewed the code. Imagine if I were to maintain the code, make some changes here and there. I would have been frustrated to get the output and worse is, it would have taken a lot of time.

Lesson Learnt: If I write bad code, it will start biting either me or who is going to maintain the code.

Short term gain = long-term loss

Initially I had the focus to create the working software. So even though I realized that there were places where I could have improved, I deliberately left them in the interest of time, not to deviate, or from my pure laziness. I took short-term gains, I kept on taking those short-term gains. They were nice, but in the end, I had to lose in the long-term.

Lesson Learnt: I should have focused on long-term rather than short-term.

All said, I was very satisfied with the working code I had. Because that's a measure of success and a critical source of encouragement. There is no better thing than to see your own code working and producing results that you expected. But in the process, if you also take care of the code that you are writing, over a period of time, you will have 2 things to brag about. One is the working software or output and second thing is the actual Code and the way it is written.

CHAPTER 2 - HOW TO LEARN PROGRAMMING

Beginners often have this question of “How to learn programming”. This is an important question to ask and answer since things could make or break at this stage. So I will share a few points out of my own experience as well as industry recommendations.

Be passionate

To learn programming, you have to be passionate about it. You have to be passionate about creating something or building something. Don’t do it just for the sake that IT industry is hot at the moment or any other stupid reasons!! IT is not hot, IT is consolidating. IT will always grow but if you want to grow, you need to be passionate. So unless you have the passion, you cannot survive in the long term.

Program in your area of interest

This is going to make the difference. Try to program in your area of interest. Find out your area of interest. And do something in that area. After I created this Macro Assembler as part of the academic project that I described in the previous post, I went ahead and did two things.

Cricket Simulator game in C

I always loved cricket and so I started to create a cricket simulator game on my own. The journey was wonderful, because it had multiple modes such as One dayers, Test matches (Those were not the times of T20s). And I created the logic, the data, the algorithms, the strategies for the game and I coded them all. I used pointers, I used File system, String parsing, Pseudo-Random generation, etc. The point was I only used them when required in the game and not before.

Artificial Intelligence Scripting for Age Of Empires (I am a huge fan)

Since by now I was fairly confident of my programming abilities, I ventured into a new proprietary Scripting framework designed for Age of Empires game. That was even more fun as you could control how the computer would play against humans.

Moral: Programming in the area of interest will make your programming adventures more enjoyable!! And this helped me to have something to brag about during my campus interview.

Practice, practice and practice

Programmers love to code. They want to create things more and more efficiently. So all they need to do is practice, practice and practice. Practice definitely makes a Programmer perfect.

Review your own code

Reviewing your own code will give you a reality check as to where you currently are and where you are headed. Some of the simple things that you can easily review are:

1. Exception/Error handling
2. Comments
3. Naming conventions
4. Code Design
5. Modularity
6. Abstraction

Ask for opinions

When you feel that you need help in guiding you, please ask someone for help. It might be your friends, or an expert, or a public forum. Posting your code in a public forum will get you lots of opinions about your code. You will get some good critics also, that will help you to improve your coding skills.

Read good programming books / blogs

Programming is like any other subject. Easy to learn, but challenging to master. So you need to read a lot of books/blogs to keep on improving yourselves and apply those learning to your project. Improve your thought process and it will automatically shape you to become a much effective programmer.

So now that you know how to get started, why don’t you start some favorite pet project right away before even trying to move to the next chapter?

CHAPTER 3 – THOUGHTFUL QUOTES FOR PROGRAMMERS

I always believe that programmers also need a lot of motivation and inspiration to achieve greatness in their programming careers. So here you go on some of my thoughts.

"Programming is a rhythm game, the more you are in rhythm, the better output you would produce"
"A refactor a day keeps technical debt away"
"A tired programmer is a stupid programmer"
"A good programmer is one who can think of many different solutions to a coding problem and applies the best solution to that problem"
"Code is like an art, the more artistic approach you take to build it, the more beautiful it will be"
"The best time to review your code is NOW, don't delay it"
"Your code is like your girlfriend, the more you take care of it, the less it will bug you"
"Code should be so expressive that comments should not be needed to understand the code"

“Bad code is like diesel engine, it has high maintenance costs. Good code is like petrol engine, it has low maintenance costs”

“Code is an art and the programmer is an artist, it all depends on the artist on how beautiful he wants his / her art to be. The more effort he puts in, the more beautiful the code will be”

“Programmer is not an individual entity, he is part of an entire software team that is doing something meaningful.”

“A good programmer will always have a desire to get better and better, to code better and better as day progresses”

“Once you understand the programming concepts, learning a new programming language is not difficult. But it takes time to master a new language”

Chapter 4 - Simple Programming Tips

Quite often it is the simple things that we often miss out while programming. That results in errors, difficulty in maintenance of the code that we write, increase in the efforts of finishing modules, delivery slippage, etc. But these can be avoided with some simple tips that follow.

Exceptions

1. Avoid Unhandled Exceptions
2. Catch errors and give proper messages to the user

Variables

3. Follow proper naming conventions
4. Always initialize variables and reset them to their initial or final state
5. Always reset variables if you are using them in loops
6. Avoid long variable names