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

7. Avoid static variables unless it is really required

Methods

8. Method names should be meaningful and convey what it is trying to do
9. Method names should not be too long
10. Avoid long methods, typically it should be only around 30-40 lines
11. Design methods for reusability, don’t have any dependencies in the method
12. Always think of plug and play methods

Parameters

13. Avoid too many parameters for a method, if so you are missing an object or a structure
14. Avoid more than one return variable, if so you are missing something

Checking

15. Check if NULL

16. Check if empty
17. Always check for array length before processing to avoid index out of bounds exception
18. Always cast properly to avoid invalid cast exception
19. Always check boundary conditions

Memory Management

20. Always dispose objects after use
21. Always have long running methods in separate thread
22. Always give a response UI to the user, it should never say “Not Responding”

General

23. Always follow this principle while programming
    Initialize -> Do the work -> Cleanup
24. Always abstract the functionalities
25. Avoid using = instead of == for equality comparison
26. Always check the interfaces for errors (Interactions between two modules)
27. Use a Version Control system which can solve major headaches

28. Always check-in your code very frequently, it can save a lot of time
29. Refactor the code very often till it becomes a habit
30. Use break statements only when needed, use them carefully
31. Database connections should be opened only when needed, and should be closed as long as it is not needed
32. Do not fetch all the records from the database, especially if the records are huge in size. Use paging concepts.
33. Have all environment specific variables in a config file, for eg. Connecting to a database server, Default file path, etc.
34. Follow proper coding standards
35. Always review your own code
36. Always read good programming books

CHAPTER 5 - ATTITUDES OF A GREAT SOFTWARE DEVELOPER

Software development is an art, not just a science. You can learn all the technicalities of software development, but you need to be absolutely passionate about coding and perceive it as an art to be extremely good at it. If you are one such person, I will introduce you to the journey of becoming a "Great Developer". The objective of a Great Developer, as i name him/her is to make his/her art as beautiful as possible and make it the best.

In my own thoughts, I will share some attitudes which a great developer should have apart from the general expectations of being technically and analytically sound, understanding requirements in detail, good design skills, etc.

Attitude #1 - A bug is a question of my ability to write good code

Fixing bugs is part and parcel of a software developer's activities. A bug is obviously the worst enemy of a Developer. But how many developers think in the following lines while fixing the defects

1. What I could have done to avoid this bug in the first place?
2. How did I allow this bug to escape my eyes?
3. OK, something wrong has happened this time. How do I avoid the same mistake next time? What steps do I need to take?

Truth is very few developers think on those lines.

A person willing to be a great developer should consider a bug as a threat to his position, as a threat to his credibility, as a threat to his programming skills. That is the attitude that will make him/her a great developer.

Attitude #2 - Mr. Tester, I challenge you to find bugs in my code

How many developers have this attitude? Many developers think that the job of the testers is to find bugs. Yes. Obviously, but that doesn't mean as developers, we can take bugs for granted.

A great developer or a person willing to be a great developer should always invite / challenge the tester to find bugs in his/her code. He should have so much confidence in his code that he can challenge in such a way.

Attitude #3 - No compromise on code quality

Code quality should be of prime importance to a developer. That will include following the right coding standards, making the code more maintainable using proper design and code refactoring, etc, etc. But how many of us compromise code quality for many reasons best known to us?

I can quote an instance in my project to explain this. I was leading a team of developers and we were working on fixing something in the very last hours of a Friday night. We had to give a build on Monday. All of us were looking into the problem. I got curious as I saw the problem and started getting my hands dirty in the code. Time went by and only the last 5 mins were left for everyone's cab. It was a make or break. We had to come the next day, if that was not solved today. I

did something at that time, which absolutely infuriated all my team members. Unable to see the clarity in the running code, I refactored a bunch of lines at that last minute. Everyone were so pissed off, that they started scolding me :-) asking if it was so important at that moment. I answered "Yes, it is that important". Of course we worked the next day for other reasons, but the whole point was even though I had an option of fixing the code in the running code, I chose to refactor the code not compromising on the code quality.

A great developer or a person willing to become a great developer should never compromise on the code quality, no matter what.

Attitude #4 - Confident but not arrogant

A great developer or a person willing to be a great developer should be absolutely confident of his abilities but should not be arrogant towards fellow developers and testers. He should always remember that he is part of a team that is working towards a common goal of shipping a project on time with good quality.

Attitude #5 - Acknowledge the Tester

It can happen that despite all the hard work and efforts put in by the great developer, a great tester can still find defects in his code. In those cases, acknowledge the great tester.

A great developer or a person willing to be a great developer should always acknowledge the tester for the bug that he found. He/she should remember that the bug is the enemy, and not the tester :-)

It is quite easy for anyone to be a developer, but takes something special to be a Great Developer. But the journey to be great often starts with simple changes in the attitude.

CHAPTER 6 - SKILLS FOR A GREAT SOFTWARE DEVELOPER

Strong in basics

If you ever want to become a great software developer, you would want to be absolutely strong in your basics. So what do I mean by basics here. Operating system fundamentals, Architecture differences, Difference between compiler and interpreter, how a program gets compiled, Difference between early and late binding, how internet works, how databases work, etc. Trust me, if you want to become great, you need to start from the basics.

Technical thinking

Technical thinking is the ability to think the details inside a complex black box. For example, how a mobile application that you use daily, actually works inside. It may not be always right, but you need to think that way in order to improve your technical thinking.

Problem solving skills

This is one of the most important skills to become a software developer, forget about becoming great. As a developer, you will be solving problems all the time. Be it simple ones or highly complex ones, a developer needs to be good at problem solving. One small tip here would be to break bigger problems into smaller and easily solvable ones. It feels so good if you can attack the smaller problems and then connect them together to see how you have solved a complex problem.

Algorithms and Data Structures

One definite thing that can make a difference between an average developer and a great developer is his/her knowledge on Data Structures and Algorithms and how it gets applied. Since you will be solving the simple and complicated of the problems, you need to be aware, how to efficiently solve each and every problem. That is where the knowledge of data structures and algorithms really helps.

Design Patterns

This one can drastically reduce the code's maintenance effort. Design patterns are solutions to common coding problems. They ease the pressure of maintenance by following a standardized approach, allowing flexibility yet modification of code. Thus it is imperative that a developer gets good at using appropriate Design Patterns in the right places

Team work

A great developer is almost always a great team player. He/she is willing to help others learn the art of programming and also willing to work as a team.

Attention to detail

This comes as part of the technical thinking. A developer gives a lot of attention to details, as details are very important to the overall fine tuning of the application.

Long term thinking

A great developer doesn't go for Quick Fixes or Hot Fixes. He might occasionally do it, but his main intent is to provide a long term solution to any problem.

Communication skills

A great developer needs to have good communication skills. His/her role is not just to program but to grasp the problems from the customers, discuss & brainstorm with the team, come up with the best solution, convince the management about the efforts, talk to the customers, present the solution, accommodating changes from customers, etc. Hence it is imperative that the developer needs to improve on the communication skills.

Customer Focus

Every project starts with a customer requirement. A great developer starts with the customer requirements/pain points and works towards a solution that will solve the problem for the customer. By doing so, not only that he is developing good code, but he is developing something that the customers actually enjoy.

Continuous Learning

A great developer always loves to learn something new in his/her art. It is either through peers, websites, blogs, by contributing to open source projects, attending conferences, etc. He is always in touch with the latest trends

CHAPTER 7 - TEAM WORK IN PROGRAMMING

Good programmers are good at their skill, i.e. Programming. But is that enough? Will their programming skills alone suffice to do the job. Unfortunately no. A programmer invariably is going to be part of a team, maybe small or large. So it is important for a programmer to understand the importance of team work especially in the context of programming. I am listing a few points that might help you in understanding its importance

Good team is more powerful than individuals

Any programmer has to understand that the output of a team is going to be more than the sum of their individual outputs. That is why he should be more inclined to work as a team rather than an individual.

Customer is in the team

One important thing to note here is that the team is not just filled with developers and testers, It actually contains the most important member of the team – The Customer. The Customer is the starting point of all requirements, so listening to him/her is the best thing a programmer can do. Many programmers tend to think of what's more easier for themselves rather than thinking what makes it easier for customers. I have also been doing that, but luckily I changed those, once I started thinking from the customer's perspective. Then I would think of the technical aspects such as feasibility. Remember, if Customer is happy you get more business, you have coding work to do for more time ☺

Project manager is in the team

Another important member of the team probably is the project manager. A developer needs to be in good rapport with Project Managers. Developers often always feel that Project Managers are actually rogues who don't care for employees. Most of the time, this feeling is true, but the point here is Project Managers are assigned a crucial task of getting the project delivered on time, on budget and

with good quality. And generally they will get to choose only 2 of the 3. Project Managers tend to take tough decisions at times due to these constraints. Having a good relation with PM will go a long way in making him understand the technical feasibility, efforts, concepts of how quick fixes affect the quality, etc.

Tester is in the team

Arguably the number one friend (Read Enemy ☺) for a developer/programmer is the Tester. Remember one thing, Developers and Testers work towards a common goal. That is to ship the product with high quality, and a tester ensures that you maintain the quality. So feedback from the testers should be highly valued and respected, because invariably they think from the end user or customer perspective. Maintaining a friendly relation with tester goes a long way in maintaining good team morale and thereby improves the overall quality of the product/software under development.

Pair Programming provides instant feedback

Most of the people find it hard to do their work if someone is around. Even if it is their close friend. Everyone is inclined towards their privacy and freedom. That is good in a sense, but when it comes to coding, having 2 people develop the same code really helps in improving the code quality. This is called Pair Programming where 2 people sit together and they begin to solve the coding problem. If one person codes, the other person reviews, critiques and provides suggestions for improving the code, algorithm and everything thereby providing instant feedback. And they reverse roles after sometime. This is one of the proven methodologies that works in Software Development and beginners should understand its importance and should not consider it as a question to their skillsets, or as a threat. Rather they should think it as an opportunity to become better at their art.

Listen to everybody

One of the most important traits is to listen to everybody, the customer, the project manager, the project lead, the testers, the Business Analyst and all the roles that the current project will contain. Everyone has some point to make from their perspective. You need not necessarily agree to each of those points. You are always open to debate, but you need to actively listen to them and then put forward your points.

CHAPTER 8 – THE ART OF DEBUGGING

In software, everyone faces issues almost daily. We need to be highly equipped to solve those issues. Not all problems are straightforward to find out and require a lot of debugging to be done. Hence debugging skills is of prime importance. I would like to reiterate that everything in software development is an art. Similarly debugging is also an art. In this chapter I will focus on how to debug a software problem.

Understand the architecture

The more the number of components in a system, the more complex it gets to solve an issue. It is critical to understand the architecture of the software / the module in order to debug a problem. Suppose your system has a client web browser and at the server side a web server

and a database server. What if this setup is required to be able to work through a VPN connection? And you get an error in your web server mentioning that you "Could not connect to the database server", where do you think the problem might be. The problem might be anywhere. Hence it is very important to understand the architecture if you want to solve the problem quickly.

Analyze the entire picture

Once you get the entire picture, analyze the entire picture and try to see where the symptoms are lying. I refer symptoms here because whatever you see is only a symptom, we need to trace to the problem in order to solve it.

From the symptom, try to trace the problem

Many problems tend to have more than one symptom, look out for those and try to get those hints. In our example, "Could not connect to the database" is a symptom. The problem might be due to various reasons.

Try to list different possibilities

Try to list out the different possibilities that might result in the actual symptom. In our example, the symptom could be due to the following reasons

- Database server is down
- Database server is not listening on the expected port
- Database server is not accessible from the web server
- Database server is not accessible through the VPN
- Database server port is not accessible through VPN

Make an educated guess where the problem might be

Out of the possibilities, try to make an educated guess where the problem might be and try to dig into the problem.

Eliminate component by component

If you can't make an educated guess, you need to identify the problem with the help of an elimination round. Since you have the basic

understanding of the architecture, try to eliminate it component by component. That is how complex problems are solved by breaking them into individual pieces.

Debugging is often an underrated skill in the world of Software Development. However it is one of the critical skills that we need in order to survive or shine in the software industry.

CONTACT INFORMATION

If you have any feedback about this E-Book please send it to Rajaraman.raghuraman@gmail.com

Facebook: https://www.facebook.com/rajaraman.raghuraman.5

Twitter: https://twitter.com/raja4tech

GooglePlus: https://plus.google.com/100297834218867757772

Blog for Professionals: http://agiledevtest.blogspot.in

STOP DREAMING AND BECOME AN AUTHOR YOURSELF TODAY!

It's Free, Easy and Fun!

At our sister website, Foboko.com, we provide you with a free 'Social Publishing Wizard' which guides you every step of the eBook creation/writing process and let's your friends or the entire community help along the way!

LOGON ONTO FOBOKO.COM

and get your story told!

FOBOKO.COM