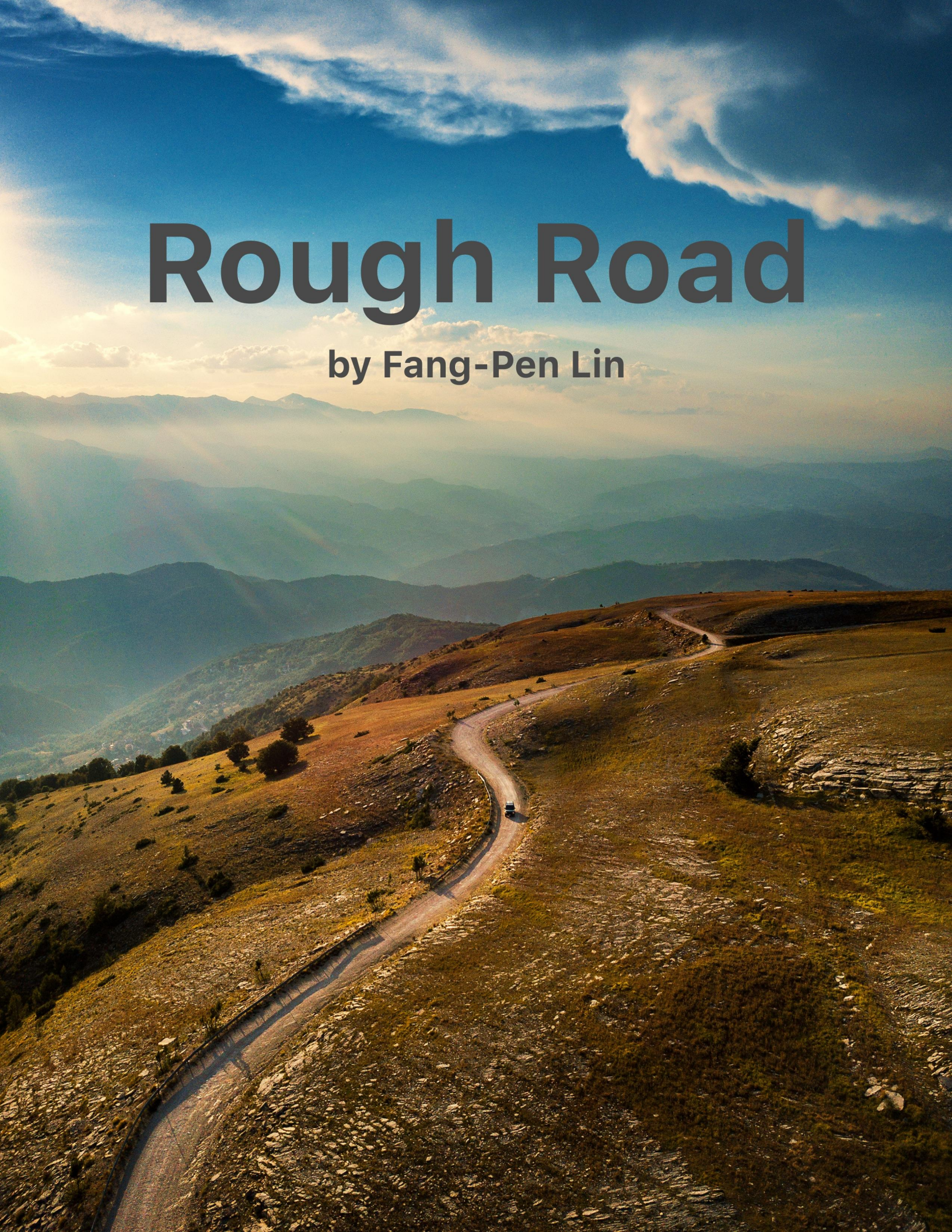
Rough Road
by Fang-Pen Lin

Rough Road

A guide for building software engineer career from ground up to top level based on twenty years experience

Fang-Pen Lin

This book is for sale at http://leanpub.com/rough-road

This version was published on 2023-08-16

Contents

READ ME BEFORE READING THE BOOK

Hi there, thanks for reading this book!

As you may notice, this book is still in its early stage of writing. I am adopting lean publishing model[1] here. Which means this book is published when it's not complete and sold at lower price at very first. Over time, when I add more content to the book, and I will raise the price correspondingly. For the readers who bought this book early, you can enjoy your early bird discount while there's more content added later on.

Please notice that the outline or even the content of this book you see right now doesn't necessary mean they will stay in the final result. There are content, structure or direction, or even the writing style yet to be figured out. For example, I think there are some skills are also important for being a great software engineer, but I haven't figured out which chapter it belongs to, so it's not even listed here yet.

Feedbacks are welcome, since the whole point of leanpub publishing model is to collect feedbacks early on. Please also feel free to let me know if there's any topic you want me to cover. You can reach me at

hello@fangpenlin.com

Enjoy your read :)

[1] https://leanpub.com/lean

Preface

Software is eating the world, and the force behind this trend is software engineers. As the software jobs demand surge, more and more people want to ride on this once in a lifetime wave. Programming is not that hard to learn, but it's challenging to master. Just like many people can swim doesn't necessarily mean they can all swim as fast as Olympics champions. As you can imagine, it takes years after years training and practice to be an Olympics champion, so as a topnotch software engineer. However, given the high demand for engineers, you can find plenty of software books or courses out there claiming absurd things such as mastering programming in a few weeks. Surely if the goal is to learn fundamental programming skills and find a job in the tech industry, yes it's possible. But to master it, certainly not.

Like all different kind of skills, programming takes if not decades but many years to master. And yet to be a great software engineer, many skills other than programming are also required. It's a marathon rather than a sprint. When we look at the history of sport world records, the numbers were renewed year after year. Is it because human beings evolved in these years to be able to run faster and jump higher? No, the key is knowledge. People learn the experience from the past, think about what can be improved, such as the way we train, or the tool we use. The same applies to being an engineer. While there's no lack of books teaching you specific programming languages, frameworks and project management, but what about learning to program and building the career as a whole from the ground up? There aren't many books talking about this aspect, be it what programming language to learn, how to develop and practice your skills, how to find your dream job or even create one your own.

As a programmer, I was lucky to find out what I love to do in my life at a very young age and dedicated almost every single day in the past twenty years to programming. To give you a sense of what it looks like, here's my GitHub commit history in the recent ten years:

My GitHub commit history in the last ten years

These are just the only records available on GitHub. Twenty years is not a short time, assume I spend 5 hours a day in programming, twenty years means 36,500 hours spent on it so far. It's not an exaggeration to say that I dedicated my life in programming.

When I look back at the years of a lifetime spent on learning programming, the hard way by teaching myself everything, I recall this is very challenging. But I never felt regret that I took this route since it's also rewarding at the same time. I enjoy the journey and would do it again without a doubt if I can make a choose another time. However, from time to time, I wonder, wouldn't it be nice if there was someone who could guide me. Even if it just helps to move the progress forward say one year in the long run. Given how much money a software engineer can make in a year nowadays, it's tremendous. So far, I didn't find a mentor in my life, but yet I still came pretty far by myself. With the experience I gain in the two decades, I want to write a book I wish I had twenty years ago, for you, like me, who enjoys programming and thinks about dedicating their time pursuing to be not just one, but a great one. Brace yourself, it's going to be a rough road, but the view is going to be beautiful.

Updates

- 2022-04-30: Added update chapter to help reader keep track what's added
- 2022-04-30: Added Meta-learning Ecosystem and Portability section
- 2023-08-16: Added HIIT section in Practice chapter

Motivations

To be a great software engineer, finding your own motivation is critical. It's never easy to be great at anything, and it may take you a decade or even longer time to get there. Skill development is always a marathon rather than a sprint. Without knowing your own motivations, you could run out of energy quickly and don't know how to recharge yourself. There are many challenging barriers to overcome on the road, you will probably end up burned out and giving up at some point. Your own motivation is like a compass, it will guide you when you are lost. Motivation doesn't necessarily need to be something huge like you want to change the world, it could be as simple as you just enjoying it or you wish to build a game at very first. From time to time, maybe it's not that obvious at the very beginning why you want this, it takes time to develop the interests and explore yourself to know that if this is the right thing for you. Don't give up too quickly, if you felt coding is boring or too hard, maybe it's just because you don't know what you can do with it yet, or perhaps your school forces you to learn it and they didn't do a good job showing how fun and useful it is. Now, let's talks about the common motivations people have, how you can find it and share some of my thoughts.

Code is magic

Have you ever watched a Harry Potter movie or read the book? Have you ever wondered, wouldn't it be nice to be able to use magic in real life, control things with a wand and spells? Yes, you can. Thanks to modern technology, nowadays, code is magic, it can literally control so many things in our world with just pure language, from electric vehicles to nuclear plants. It is the closest thing to magic. The beauty of programming is, you can extend your will to a machine, and let it does things for you. The aha moment is one of the most satisfying parts of programming. When you see the code finally runs as your command, you feel powerful, in control, you're the creator who gives life to the machine. In other words, you simply enjoy the sense of accomplishments coding brings you. Congratulations! Not all people truly enjoy programming as their job, and when they do, they usually push themselves forward without anybody asking, and end up being a great engineer. It's one of the best kind of motivations you can have.

However, it's actually very hard to even get to this aha moment where you feel powerful and in control of things in the first place. Usually, it's very frustrating at the very beginning as nothing works as expected. This is totally normal. As I emphasized, this is a marathon, you need to be patient, the beautiful view will come after the uphill.

Build something you want

Like many people, I taught myself programming since I want to build a game. When I was a kid, there was an MMORPG that is very popular among my classmates. I really like this game, but the game was very hardcore, they made it insanely hard to reach the top level in the game. As a student, I had very little time to play, and the internet was also still pretty expensive by the time, there's no way for me to get to the top level. Then I wondered, well, if I cannot be great at the game, why not make my own? In my own game, I can be the god, I can do anything I want by simply typing a command. When I recall, this doesn't sound creepy at all until I watched a TV show on Netflix. Anyway, building something you want could be a good motivation, since you have a very clear goal you want to achieve, as long as the desire is strong enough and persists, it can drive you moving forward.

Screenshot from my MMORPG project's game client

However, the things you want to build could be very hard, this is totally normal. Like mine is to build an MMORPG. Usually, it takes a whole team of experienced software engineers spends maybe a few years to create one, needless to say, how hard it is. In the beginning, it was too hard even to make any progress, so I detoured a bit to build something easier first. When I learned something new from the more straightforward projects, I felt excited and can't wait to apply on the major project. Then made some progress and hit a wall again, detour and learn new tricks, then come back later.

Actually, I didn't manage to finish the game but at least made significant progress, and I've learned so much from doing it. We will talk more about how you learn from building things in the learning section later on. At very first, programming is just a tool you use to get what you want. Later on, you may find you actually enjoy it, just like I did. And remember, the beauty of software, as long as in theory, everything makes sense, then it's totally possible to build it. It's just a matter of time. So if you don't give up and persist, you can make it one day even if you're only one person along. Yes, it looks like impossible at first glance, but you can always detour, for now, learn new things and come back later. It's like playing RPG games, and in this game you know you can challenge the final boss right now just get out of the newbie village, you tried and failed epically. You can go somewhere else to level up and come back later to try again.

The thing you want to build doesn't necessarily need to be ambitious, it can be as simple as a tool helps to make your life easier. Personally, I keep a to-do-list of things I want to build whenever I thought of anything interesting, now there are hundreds of them. For me, thanks to the list, I never worry about what to make, the only problem is that I only have 24 hours a day. Always having the desire to build things is very helpful for fueling your motivation. Some people may say I don't know what to build, how can I come up with ideas? To me, these ideas usually come out of daily life, there are actually everywhere. Just you need to observe carefully and think about how things work. Then you will notice there's gotta be a better way of doing things, especially with help from software. This is not easy at first, and no shortcut to achieving, it's call first principles thinking[2]. To practice, in your daily life, you can keep thinking about things like this:

- Don't assume what exists in our world is always right
- Why are they making it works like this?
- Is there any better way to do it?
- How would you design it instead if you were in charge?
- If this is better, why aren't people doing it already? What's the challenge?

As long as you can practice first principles thinking, you will find out that there are many exciting ideas to explore. And don't be afraid of any idea comes out of your mind you want to build to be stupid or not a great idea. You may hear people saying, or even your own voice in the head:

- You are never going to make it
- This is not going to work because of xyz
- This sounds stupid
- There's already something like this
- How are you going to get people to use it?
- What's the business model?

I would suggest ignoring them and build it anyway. Since in the end, even if you fail, you still learn a lot from it. And the things you learn is your asset nobody can steal it. If you give up just because

[2] https://jamesclear.com/first-principles

people saying something, you won't get a chance to learn. Even if your idea could be not so perfect, after you trying hard to build it, you will learn what could be wrong. There are countless examples in Silicon Valley, people have ideas, and they got laughed at, and they didn't give up because of that, and keep doing it.

A great example could be Elon Musk's dream of reusable rockets. It might be laughable when he proposed the ideas, but not so much after people seeing two rockets land themselves perfectly[3]. So remember, it's not the people who only know how to laugh at others push the world forward, it's the builders like you.

Grit

Not that many people love to take on a challenge, and if you do, you're the lucky ones. It's definitely very challenging to become a top software engineer, so if you love challenges, you will have a better chance to become one. However, just enjoy taking challenges from time to time is not quite enough, you need to persist. This kind of characteristics is grit. I recommend watching this talk Grit: The Power of Passion and Perseverance[4] by Angela Lee Duckworth. You will see that it's not the people with higher IQ tend to be successful, it's the people who keep pushing themselves and never gives up. I saw similar things in the software industry.

In the past, I met some great software engineers who used to be very top-notch in other fields. These people are

- Top class pianist
- World champion of sport
- Elite athlete
- ...

And it's surprising to me considering how much time these people spent on a non-relative area previously, they are doing pretty well compares to many people who's major is Computer Science and worked in the industry for many years. Sometimes I even make fun of ourselves with my co-workers saying that if these top talents from other fields all come to the tech industry, then we are all going to lose our job. Joke aside, you will notice that these people share a common characteristic, and guess what, it's grit.

In the end, what's grit you ask, to me, I think it is

- Believe you can do it as long as you really want to
- Enjoy taking challenges
- Always think about how to do better
- Don't give up even if you fail from time to time

[3] https://www.youtube.com/watch?v=u0-pfzKbh2k
[4] https://www.ted.com/talks/angela_lee_duckworth_grit_the_power_of_passion_and_perseverance

You are lucky if this already who you are. Just keep pushing, with right method then it a matter of time. However as much as it's a blessing, it's also a curse in the same time. Be careful not to push yourself too hard. When you realize that you are coming close to the edge, just give yourself a break. And again, think it as a marathon, remember, you're not giving up, just taking a break for the long run..

Desire to learn

If you are a junkie of learning, you are in the right business. One benefit of being a software engineer is, there's always no lack of new things to learn. Thanks to the internet, given how easy it is to spread source code and software, to exchange ideas, the innovation speed of the software world is astonishing. This year you've learned this new framework, next year it becomes old and out-dated. Don't be surprised, your whole engineer career is actually all about learning. Usually, this is not the kind of learn once and apply entire life job. However, you know what, you can still find engineering jobs that only require you to learn certain things and rarely need to update if you want. But if that's your goal, and you see the opportunity of learning new things as a downside, you probably just bought a wrong book.

Learning a new thing could always feel like kids in Christmas eve oping a present box see what's inside. That's an excellent motivation for building your software engineer career in the long run. Compares to others, this is probably one of the most essential motivations of being a great engineer. Therefore, this book, to some degree, is meant to be a book of *meta-learning*, which means learning about learning. If you already enjoy learning, that's great. But keep in mind, just like eating food, people say

> You are what you eat

For software engineers, I would say

> You are what you learn

What you've learned is like food for your mind, when you eat healthy food you grow stronger, when you eat junk food you might become ill. So it's also pretty important as a software engineer to pick what to learn. How you learn it also matters, there are many tips to help you learn faster and better. We will cover this topic more in the learning section as we mention many times, it's all about learning.

We talked about the cases about you're already interested in learning new things, what if you're not? Don't blame yourself so fast, that probably may not be your fault. Nowadays, the education system we have was designed a long time ago for industry needs. They are expected to teach you some skills, and you can apply in your factory job for the entire life. Things have changed recently, but still, the system hasn't kept up. In the end, they only expect you to learn something in the school, rather than figure out how to learn by yourself. So here's what they always say

> It's your job to learn as a student

Which is obviously wrong given what now and future society needs is

> It's your job to learn meta-learning as a student

Since meta-learning is not taught in the current system, it won't be surprising people don't know what's the fun of learning itself when you are just forced to learn something you don't know what they are for. I think in the end, people don't see the purpose of why they need to learn these things is the key. Remember what we talked about the motivation of building what you want previously. With the goal of building something you want in mind, and you encounter problems, and you learn how to solve them, this is a more natural way to grow interests in learning. So I would suggest, forget about learning, as if it sounds boring already to you, focus on building something or other goals you want to achieve. As long as you keep solving problems, you will soon realize learning new things is great as now the issue can be resolved quickly, then you know why you are learning it, and you will want more.

Money

In Silicon Valley, the salary of a junior software engineer can easily be more than 100,000 USD per year as I am writing this book. Considering the cost of living, it won't be too cozy, but it's still a pretty decent level of salary compares to most of the non-engineer jobs. The same could also apply to other areas, given the high demand, software engineer usually paid pretty well. And you might also hear many software engineer's success stories as self-made millionaires or even billionaires. So yes, being a software engineer could make good money. And not surprisingly, that's why many people want to be a software engineer. There's nothing wrong about this, I am not here to be judgmental about money being the motivation, I think it's great if that's what pushes you forward, you should go for it. Just, however, it won't last long as if it's your only motivation, let me explain why.

First, being great is not necessary for making yourself rich as a software engineer. While there were many self-made millionaires software engineer stories, at the same time, there were actually more stories of failure you never heard of. I've seen many people who are good at building great software but fail to sell them. There are also companies doing everything right with software, but simply fail to raise more fund and shutdown. All in all, there's no guarantee being a great software engineer could make a fortune for you, and sometimes, you need to be lucky to get there. I am not saying being good at what you do is not essential for seizing the opportunity, but it's actually more like a lottery ticket.

Second, as an employee, being a ten times productive software engineer actually doesn't make you ten times salary. While this sounds unfair, it is the reality. When it comes to compensations, people compare with their peers, and they feel upset when they see one's salary is way higher than their own. Employers usually want to avoid this kind of situation. Besides, it's tough to measure the performance of an engineer objectively. It often requires good experience to tell if a software

engineer is doing a great job or not. Moreover, how much you get paid from time to time is more how you negotiate it. Many great software engineers are really good at what they do, but they don't know how to negotiate salary and end up getting paid even lesser than their co-workers. As a result, a great software engineer doesn't always get paid more even when they are more productive.

Finally, there are more ways to make money than being a great engineer. You can always make money by doing things other than coding. If you are good at selling, maybe you can be a real estate agent. If you are interested in medical, why not be a doctor? There just happens to be high demand for software engineers now, doesn't necessarily mean it's the only way to make money. Don't get me wrong, the top software engineer still makes great money, but considering the time and efforts to get there, it may not be worthwhile if money is the only reason. After all, comparing to spending countless lonely nights on improving skill, it's a known secret that negotiating hard on salary and keeping shopping around between different companies year after year is an easier way to get paid more as if money is the only goal.

While money sounds like not such a great motivation along in the long run, nevertheless, if you happen to have financial pressure and programming is the only way you know how to make money, don't shy away. Not everybody has the privilege to not to worry about money, it's still a very good reason to do it, maybe you will find other motivations along on the road. And it's actually fine if you cannot find anything else in programming, the worst case is that this just becomes a job, and that's it. The best case is if you grow love into it, and you get paid to do what you love to do every day, but you don't want to underpay yourself for your great work, so how you negotiate your salary is still pretty important. We will talk about this in the later section.

In the end, despite that we talked so much about money as the motivation down, it actually can be a great part of your many motivations in the long run. As if you are really into building things, without good financial support, you won't be able to do whatever you want, you will realize that you want to be able to create things freely without worrying about money issues. If that's your ultimate goal, yes, it makes more sense to last long along with the goal being a great software engineer. We will also cover more on how to get financial freedom later.

What kind of engineer you want to be?

In the journey of pursuing to be a great engineer, it's always helpful to understand what kind of engineer you want to be. Different type of engineer focus on different type of fields. Some engineers love security research, they study zero-day exploits, learning how to attack and defense systems. Some engineers love data, they want to be able to process millions of data efficiently and reliably. Some engineers like UI interactions, they love crafting UI down the pixel details and pursuing the best user experience. Everybody got something they like. People in the tech industry trend to divide software engineers specialized in different fields into different roles, some of the typical roles you see out there are

- Backend Engineer

- Frontend Engineer
- Mobile Engineer
- Security Engineer
- Networking Engineer
- Data Engineer
- Build Engineer
- DevOp Engineer
- QA Engineer

The list here is not complete can keep going on. While it sounds like you need to make a choice and can only do one thing for your entire career, it's actually not true. Given myself as an example, I am generally interested in many fields and have more or less experience in them during my twenty years of an intensive programming career. I know I don't want to set a boundary for what I do, so I instead of making myself as any of these, I position myself to be a Master Generalist Software Engineer. By Master Generalist Software Engineer, it means when facing an engineering problem, I usually don't care about what fields got involved, I divide the problem and care more about how can these be conquered individually. And I will learn whatever it's necessary to get there.

A great example is an open-source project called Embassy[5] I built for one of my previous employers. I wrote an article Embedded web server for iOS UI testing[6] for it. As you can see, this project is for mobile, does it make me a mobile engineer? And the purpose is testing, does it make me a QA engineer? And yet the technology I used is actually from my backend server networking side experience, does it make me a backend engineer? To be honest, I don't actually care. Because of that, this is my superpower to be able to jump into different problems and solve with solutions from different perspectives and not bound by the mobile engineer title as it was the official one at that moment.

In the real world, companies tend to hire people based on these specific roles, but as I said, it doesn't necessarily mean you need to be limited by that. The point is you know what kind of engineer you are or want to be. As long as you know, it's easier to figure out what to learn and what skill to develop. If you don't know what kind of engineer you want to be, which is fine, you just keep exploring. Say if you are doing the mobile job in your company, and there's a new project happens to need backend people's help, if you want to try backend out, you can say you want to help. Over time, the more you try, the more clear what kind of engineer you want to be. With a clear expectation for yourself, it also helps driving yourself forward in the long run.

Have fun

TODO:

[5] https://github.com/envoy/Embassy
[6] https://envoy.engineering/embedded-web-server-for-ios-ui-testing-8ff3cef513df

Treat yourself well

TODO:

Positive feedback loop

We've talked about different kind of motivations, in the end, it actually doesn't matter what type of motivation you have. Everybody got their own story, as long as they can drive you moving forward in the long run, they are good motivations. In a different stage of your life, your motivation could change, which is totally normal. And no matter what, to get far, you need to create a positive feedback loop. A positive feedback loop here means, when you learn programming and work on it, during the process, you gain a lot of positive feelings, be it confidence or sense of accomplishment, as long as it's positive. These positive feelings will further encourage you to do more programming. And it will end up like a self-enforcing spinning wheel.

Positive feedback loop

This is the most critical component of building the engine for driving your career forward. Many people won't be able to push themselves far is because they failed to form a positive feedback loop, or even worse, they form a negative one. When during the process of learning and exercising programming, they let frustrating, self-doubt or any other negative feelings take over themselves. As a result, programming only makes them painful. They will be afraid to do any more of this. This will stop the wheel from spinning, and they end up going nowhere.

Am I in a negative feedback loop?

TODO:

Break out the negative feedback loop

TODO:

Meta-learning

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Learn by building

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Learning is an investment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Don’t be afraid of making mistakes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Learn once and apply everywhere

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Cairo and HTML5 canvas

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Deferred, Future and Promise

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

From inlineCallbacks to async and await

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Learn beautiful things

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Grow your sense of good and bad

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Oh my! There are too many choices

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

How do you pick what to learn?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

What's your need?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Look at the community

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Portability

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Licenses

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

The learning curve

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Pros and cons

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Who is using it?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Is the development still active?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Newer is always better?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

What about the future?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Just pick one you like

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

The optimal path to learn

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

What about university? useful or not?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Open source projects?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Pencil and paper are your friends

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Always ask why

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

The Building Blocks model

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

What can I use it for?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Grow your building block index

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Sell me your interplanetary database

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

What's the limtiations?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

What else I can use?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Build a small project with it

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Making your own

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Dig the rabbit hole

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Don’t trust book blindly

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Be open minded

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Do I need to learn math?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Teaching is learning

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

How to search for the answer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

How to ask a question online

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Practice

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Deliberate practice

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Write a tons of code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Read a tons of code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Disciplines

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Build pet projects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Long life time project, see your own consequence

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Dig the rabbit hole

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Don’t be a leetcode machine

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Get outsourcing projects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Learning project vs production project

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

HIIT

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Why HIIT worked for me?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Intensive code writing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Huge project

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Forget your code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Take a break to absorb what you've learned.

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Own the consequences

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Is HIIT for me?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Tick tock

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Fundamental skills

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Version control

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Write automatic test cases

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

SQL

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Use debugger

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Design Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Functional Programming

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Reactive Functional Programming

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Regular Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Docker

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Kubernetes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Vim or Emacs

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Best practices

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Coding style

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Technical debt awareness

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Write tests

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Review your own code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Automate the repeating tasks

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Setup CI and CD early on

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Write documents

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Write comments

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Write logs

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Polish your tool

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Think about security

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Think about a bus runs over you

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Great engineers reuse code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Career

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Build your portfolio

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Improve your writing skill

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Write blog

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

What makes a popular open source project?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Yourself as a brand

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

How to find a Job

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

This is a game of number

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Recruiters

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Negotiate salary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

How to write your resume

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

How to interview

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Big company vs small company

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Comfort zone risk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Tiny scope risk

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Coding with people is different

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Code review

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Teamwork

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Bad co-workers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Dilemma of a good engineer

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Build your own startup

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.

Miscellaneous

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/rough-road.