Being Craftsman: A Software Developers Handbook.

-Harshvardhan Kelkar

INTRODUCTION

I thought that it would have been useful for me to have a handbook geared towards new graduates when I was starting out as a Software Craftsman. This chronicles my journey over the last five years and the lessons that I learned. I hope you find then useful. Do send me a note at hershvd@gmail.com with your thoughts.

CHAPTER ONE
Beginnings

I was fresh out of graduate school having secured a master's in computer science. A master's program certainly made me delusional as far as the job prospects were concerned. I strongly believed that seeing my credentials as someone who could do research getting a job would be a piece of cake. This was towards the end of 2009, when the effects of the recession were still in play. I ended up working for startup x doing uninspiring work. Mid 2010 as I was stumbling, for those unfamiliar perusing stumbleupon. I came across preyproject . It was something that immediately caught my attention even though the webpage was in Spanish. Running it through google translate I quickly realized this was something I cared about, A geo-tracking software for Laptop's, mac's and cell phones. Discarding all my inhibitions I mailed the developers Tom and Carlos asking if they needed any help. They were

kind enough to respond positively and I found myself involved in an Open Source Project based out of Chile. During the course of my interaction with Carlos, I became acquainted with git, Blackberry app development and most of all rekindled the spark of software development in me which was missing since high school.
I moved to California and found a new job at BMC Software. My experience with PreyProject helped me deliver at BMC,
Prey took a back seat but I was determined to deliver on that front too. Last weekend I pushed onto Github the "Lock" module for the Blackberry version of Prey. The Lock module is essentially a global UI that is triggered onto a mobile screen by a push from the server side, which disables all actions on the blackberry and can only be unlocked using a passphrase set at the server-side. Furthermore, Prey runs as a service on the device. Turning it off and then on simply means the service starts again with the device. If the lock attribute is set at the server side the Lock will be in play again!

After testing functionality and seeing it work, I was filled with Pride and Joy. Pride that I had created value. Joy that I had created beauty. I did something that I really wanted to do. I created a feature which may well be used in millions of devices. But even if it isn't that wouldn't diminish my joy one bit.

One may well say there is no financial gain out of this exercise. Pete Sampras in his autobiography says he must have hit a tennis ball at least a million times before he ever set foot on court in his first major match. Malcolm Gladwell's hypothesis in his book Outliers is to gain expertise in a field One must spend at least 10,000 hours engaged in tasks of higher complexity than the previous one. Ten years to become an expert software developer as outlined by Peter Norvig. I might have come late to the party but I sure do intend to be an expert Software Developer with time. Good Software Developers are Craftsmen or Craftswomen and what makes them so good is practice. Open Source development is a very effective way of engaging with the

best developers across the globe and creating a meaningful impact by doing something you like to do. Furthermore Open Source software is a unique repository of millions of lines of code open to inspection by anyone. Read code, write code then read some more code. So stop for a moment, Find a project you like on Github or source-forge. Join a project, Drive a project but most of all remember the time when you were a kid, when anything was possible and create something just for the Joy of it. To conclude, Learn demonstrable skills beyond the computer science curriculum during your stay in school. If you are wondering what is worth learning? Read on.

CHAPTER TWO
Fundamentals

Once you put yourself on the path of radical self-improvement you start looking at the fundamentals. For me it was the realization that I couldn't touch type. Having grown-up in the era of instant messaging, I learned typing the wrong way. While I was a fast typist my finger placement was completely random. I did not rely on the home row that the keyboard provides. Bad techniques are hard to eliminate once imbibed. Now this is a generic enough skill that you should learn even if you aren't a programmer given that you will type something.

To remedy this I started working through drills on gtypist. Gtypist is a shell based utility that teaches you to type the proper way. Ratatype is a web based typing tutor that works well too. This was time well invested as now I can indeed touch-type and spend almost no time looking at the keyboard. Once you start learning your tools you make use of the cues that they

provide. In case of a standard qwerty keyboard I discovered the home row and the raised bars on the f and j keys. Those two bars alone give your fingers the entire map of the keyboard. This also highlights the need to slow down, be mindful and make use of the full potential that your tools offer. If I were to start all over again I would learn to type first. Speaking of tools, who do you think wields tools? What does the word craftsman evoke? For me it is images of toolmakers building watches, shipbuilders working in tandem towards building massive vessels and of course developers crafting code.

Progressing as a craftsman is a hard task. I think there are three key essentials for success.

1. The Guidance of Master Craftsmen or Women, henceforth referred to craftsman for the rest of this essay.
2. The burning desire to build, tweak and continuously refine.
3. The tenacity to work hard through the not so interesting parts.

Self learning is a good thing, but in the

company of master craftsmen one can make giant leaps. I rediscovered, working with the best evokes the desire to build better. There are techniques one can pick up by observation and osmosis.

I serendipitously came across a documentary called "Jiro Dreams of Sushi" Jiro is a master Sushi chef who runs a Michelin 3 star restaurant in a Tokyo Subway. Earning 3 stars from Michelin means that the place is so good that it is worthwhile to visit the country just for that restaurant! Jiro at 85 years plus is someone who is at the peak of his craft and still aims to surpass it each day.

What resonated strongly was the idea that to achieve success you need to be so good that you can't be ignored. Further building on this is another idea that you need to satisfy your work rather than rooting for some dream passion. Satisfying the work that is given to you is a stepping stone towards building expertise and thus uncovering a deeper satisfaction of the work that you do. It is enlightening to see the tough regimen that aspiring sushi

apprentices have to undergo before they can even handle fish often running into many years. Jiro's son Yoshikazu a master Sushi Chef in his own right at 50 years still gets critiqued by his demanding father. The other takeaway was simplicity and the ability to develop an expertise in a niche area. There are no shortcuts and one has to commit to a lifetime of learning. Finding a master is not easy. Why should anyone spend time to critique you? Ultimately one needs to offer something of value. Finding a way to lighten your potential master's load or solving a problem for him is one way to form a symbiotic relationship. The other motivation comes from great craftsmen themselves who want the craft to progress. In an environment like a Software company or even a Sushi bar like Jiro's it is imperative to train apprentices to take upon future roles as craftsmen themselves. This is pure economics at play once the enterprise needs to run at scale.

There are times when apprentices get assigned dreary tasks. The most successful craftsmen have gone through this phase,

tenaciously completing the assigned tasks, learning the domain along the way. Many problems are apparent only after doing the grunt work first. So do not be surprised if the first tasks you are assigned seem trivial whether it is your first job ever or you are starting out at a new one. The projects you are given are a way for your employers to assess you. Do them well. Observe what problems are being solved and what aren't? Are there processes that can be improved? Be proactive in sharing ideas after you have an understanding of your work environment. Once you can demonstrate excellent delivery of the easy work will you be given the harder stuff.

CHAPTER THREE
Tinkering

A lot of what you learn will depend on the things you tinker with. I decided to learn python. Going through the usual motions of starting with variables to moving to loops, something did not seem right.
This exercise was not fun. So to make it interesting I decided to build a tictactoe game in Python. The rest followed. It was hard no doubt having to learn new paradigms, but having a clear goal in mind helped in gathering the relevant information. Learning by doing is certainly the best way in my opinion. Yet, that is not the way we think natural. How did you come to speak the language that you do today? Did you learn by first learning the theoretical basis and the grammar or by listening and being absorbed in an environment that literally leaked the

language also known as your family.
Building a simple game like TicTacToe taught me valuable lessons.
1. Rome wasn't built in a day, the same goes for software.
2. Gaining skills takes time, be patient.
3. There is great joy to be gained from building something from scratch and that in itself is reason enough to do it.
4. This is a form of deliberate practice. Pushing yourself beyond what you can do currently is deliberate practice. Building something that you haven't yet will invariably involve some aspect that will push your limits. Keep a notebook with things you want to build next. These projects will help you expand your horizons and make you more valuable.

When you tinker, things will invariably break. The way forward is to identify and solve the breakage. Lets see how.
It was a sunny day, I stepped in my car and started out for work. Within the first hundred feet of the car moving, I heard a muffled but regular "thup a thup a thup". I

stopped the music and it was still there, I pulled over and got out of the car. Did a quick visual, checked if my bumper was hitting the tires, saw nothing amiss and started again. The sound persisted "aargh!". I pulled over again. I saw an old man approaching and thought to myself that this was great luck. I lowered my windows and asked if he could see any obvious flaws as I would attempt to move the car slowly. Hoping for an affirmative, I was aghast when he gestured to me that he was extremely hard of hearing. I got down again and started looking at the tires again. Then there it was stuck on my front tire, some kind of a poly wrapping material. Each time the tire rotated the portion that wasn't stuck was hitting the area above the tire. I took it out and problem solved. I had just completed debugging my car.

What is Debugging? At its essence given a system with x as a known and expected behavior and y as undesirable behavior, the task of reconciling the difference and finding the root cause is debugging.

In fact debugging is very much a part of programming as it is in daily life. best advice I got out of college was "learn to debug". Ironically I got the advice in an interview that I flubbed. It is analogous to doctors figuring out what is wrong with a patient or a mechanic finding trouble with a car. The goals similar, but the objects and tools employed different. The best programmers are extremely good at debugging. Some of the techniques I've come to employ are:

1. Reduce the problem space.
2. Use print statements.
3. Use Debuggers, which was my original intent in asking the old man for help.
4. Logs: Logs give a long term state of the system when turned on. It is very easy to have them running into millions of lines which leads me to point one again.

The other use of debuggers I make is to understand the behavior of unfamiliar programs. When a reading is not sufficient, It is quite handy to spin up the debugger and step through the program seeing the state of the data that it manipulates. When

you are in a situation be it in a program or your personal life, think of the debugging model employing a program or a person for your problems.

Abraham Maslow said, “If all you have is a hammer, everything looks like a nail”.
Building quality software relies on a myriad of tools. One essential pillar of craftsmanship, software or otherwise is in the mastery of tools that you work with.
The most important and essential tool that you have is your mind. All software originates as a thought so having a clear thought process is necessary. Unfortunately we have become accustomed to google, facebook and twitter. Being mindful of what you are trying to achieve is important. There is a quote from the TV show Sherlock that has stuck with me "People fill their heads with all kinds of rubbish. And that makes it hard to get at the stuff that matters. Do you see?" Treat your mind as a garden and only let in thoughts that should be nurtured,

getting rid of weeds is essential.
We must remember for all the clouds that are now available, ultimately it is a group of computers connected to a network. Quoting Larry Ellison "Google does not run on water vapor". The choice of operating system that you go with will have an effect on how much of the underlying system you understand. Choose wisely.
There are a plethora of programming languages available and to persist with the thought that all languages are made equal is a fallacy. Using the right language for the job at hand will go a long way in building successful products. Learning a different language than one which you use on a daily basis will create new ways of thinking. For instance try writing to a file first in Java, then in Python. While I am not getting in the argument of which is better, I do want to emphasize that languages have there own strengths. To deal with data in flat files, you are missing out if you do not use the trinity of awk, sed and grep. They will allow you to craft one liners rather than equivalent programs of hundreds of lines

in a high level language. From notepad to vim to sublime text to even a IDE. The text editor is where you translate your thought to code. Structured information that can make computers do what you want it to do. Mastery of the text editor will determine how long it takes you to write code. We need to see how our code evolved. Any Software that is not one time use should be maintained in a version control system. Git is my favorite, however depending on your work environment you could end up using svn or perforce. Git has many subtleties that will be apparent with practice, know your git and never worry about losing source code. Debuggers aid understanding both the program and data flow in complex projects, though let your mind compile it first. Mastering the debugger will go a long way in solving bugs in programs that you are unfamiliar with and sometimes in familiar code too. To store and retrieve data we need databases. Databases are the foundation of the applications that we build. We have SQL and not only SQL. For structured data SQL based databases are

still golden. Since real data is not always structured, that has forced the move to NOSQL databases. Redis is the swiss-knife of data in key-value pairs. Learn about couchbase or mongodb for exposure to document based databases.

This list of tools is in no way comprehensive, Building mastery of tools is a long and arduous process. However taking tiny steps today go a long way and add up in a few years. More often than not the problem you are trying to solve will drive the tools that you will use. So try to solve as many different problems as you can, slowly but surely you will see an expanding toolkit. The path I have laid for myself is to always be learning and teach what I learn.

CHAPTER FOUR
The Apprentice

To be a master, one must first be an apprentice. In the modern world, typically your first job is where the apprenticeship begins. I would prefer working on a craft much earlier than that. But, we are limited by the systems created around us like school and university. With medieval guilds no longer around, we have to craft our own apprenticeship path. The first challenge is finding masters. Some places you will find them are Universities, Companies and Open Source projects. The software industry, dynamic as it is will not afford you the luxury of one master. People move on or you do. So be comfortable with the idea of having many masters over the course of your apprenticeship.

The big mistake I made when I was younger is to think that the golden key to opportunity is a degree from an elite college. A degree is just a tool that can

open certain doors and maybe give you a head start. So don't lose hope in case you didn't go to one. The real world only cares if you can solve a business problem.
There are in fact two keys to opportunity, curiosity and a hunger to learn. Use your curiosity to learn things that you like. Don't panic if things don't make sense, everyone begins somewhere. Learn from books and the internet, the greatest learning tool of our time. Teach what you learn. Leverage what you have learned to get in a company that aligns with your curiosity. This creates a feedback mechanism for you. Learn from everyone, your peers, your superiors and even the interns.
To be a software craftsman you must learn, all the time. Do you have the hunger to learn? If you don't stop and do something else, otherwise cultivate it.

What is worth learning? Two things
1.) To satisfy the work that is given to you, you naturally must learn about it.
2.) Follow your curiosity.

That's it by learning the things needed to accomplish your tasks you immediately have become valuable to your employer.

The second thing to do is to follow your curiosity. This comes with a caveat of whittling down your interests to a few things rather than many. That will allow you to get better at your chosen interests.
Then let the magic begin, with the right people around you will learn exponentially faster and have a meaningful apprenticeship. In the path to mastery, the journey is the destination.

CHAPTER FIVE
Tests & Checklists

One of the most important ideas I encountered in building quality software is test-driven development. In TDD the mantra is red, green, refactor. Write a failing test and run it (red), make the test pass (green), then refactor it ,that is look at the code and see if you can make it any better.

If you embrace TDD you will generate code that can be shipped with confidence and will lay the groundwork to effectively communicate the code's intent in the future. With TDD you will eliminate magic from the code your write and make it deterministic and dependable.

The other tool to use while performing repetitive processes such as deploying your code to a production environment is a Checklist . Checklists are simple actions that should be ticked off before taking a critical action.

Eg.

- Check the code version
- Check that code compiles
- Check that all unit tests pass
- Check that all integration tests pass.

Only when all items on a checklist are satisfied should you go ahead with the production push. Combining Test driven development with checklists is something that all developers should consider.

One of the most notorious cases of bad code being deployed is that of Knight Capital which lost over 400 Million dollars in a Span of 30 minutes. A checklist based process would have prevented old code from being run on a single server across a group of eight.

Checklists are universally applied by airline pilots, surgeons etc. So make the checklist a part of your life. They will prevent mistakes that could happen otherwise.

CHAPTER SIX
Randomness

What makes life interesting is not predictability but randomness, I will give an example, On Superbowl sunday day I met couple of friends at Crepevine in Palo Alto for brunch. While going back I took a wrong exit and ended up in the Stanford Campus. Now the roundabout there is almost a mile inside. Upon taking the U turn I noticed a museum midway that I had not on my earlier visits there. I pulled in and it turned out it was a museum established by none other than the Stanford family itself. I parked my car and went inside to see one of the best collection of artifacts and art that I have ever seen. Added bonus that entry and parking was free. So a wrong exit made my day richer. Chance has a greater role in life than we like to admit. Relating randomness to the Software world may seem strange but bear with me. The idea I seek to present is that random explorations may lead to serendipitous occurrences. Exploring

things that you may not have encountered before goes a long way towards developing skills or adding new tools to your toolkit. For instance if you are an imperative style programmer, learning a functional language like Erlang will expose you to a different style of thinking. If you work with Mysql then learning about a NoSql database like Redis will expose you to how unstructured data can be handled. It is of course difficult to predict how these random walks may be of immediate benefit but I think the real value lies in relating the disparate knowledge to a common thread. I think randomness is also what makes the Silicon Valley an interesting place to be. If you are a technology professional in the bay area chance encounters can lead to new opportunities or even just expose you to another dimension of the technology industry that you didn't know existed.

One of the best ways to introduce randomness to your surfing habits is Stumbleupon, you can think of it as a remote control to the Internet which will recommend webpages based upon your

interests. I came across PreyProject via stumbleupon where I went on to make my first ever open-source contributions which led me to eventually work at my first big corporation job. The other source for randomness I rely upon is hacker news. In real life make use of meetup.com or your favorite programming language's user group. I think in case of real life the randomness begins once you show up which is not limited to showing up physically but also taking up projects that you care about. You like the girl you just met? Ask her out. I am engaged now so that avenue is now closed. You want to write that killer app? Start working on it then, only then can randomness help you.

Following my embrace randomness mantra, I first showed up to a talk at the Bay Area Python User Group. At the end of the talk they were looking for speakers for the next set of talks. I randomly picked "nltk" which is a natural language processing package and volunteered to talk about it. I knew nothing about it when I

first decided on the topic. Committing to give the talk forced me to learn everything about it and the end result was a very successful talk.

The second problem was that while I did give the talk the videographer did not show up, so I then decided to record the talk myself via google hangouts. The Baypiggies group now has it on their youtube channel titled "Document similarity using nltk" .

The charter I have laid down for myself is to embrace more randomness by showing up viz. building things I care about thus encountering problems that I haven't seen before and in life in general. While I cannot vouch for all outcomes, I certainly hope it will be an interesting journey.

CHAPTER SEVEN
Focus

For a software craftsman, focus is essential to build quality software. Interrupts are the enemy of focus. While not all interrupts are avoidable like meetings, interviews or even a colleague with a question. There are certain things that can be controlled like your inbox and distractions like news, facebook, twitter or any other form of content readily available. The question is how? A friend suggested a system which has worked great for me. Break down work in chunks of 25 minutes before you take a break. I use this nifty web application called moosti that helps me keep time. Ultimately it is a mental hack that allows me to give permission to myself to be present and to focus on the task at hand. With the timer counting down I tend to close all other tabs. The added benefit of this system is it allows me to measure how productive my day was. I need to count the number of successful chunks.

CHAPTER EIGHT
Career

To make a career move as a new Software Craftsman there are two questions you definitely need to ask:

1. Is it your time to learn or time to earn?
2. what are you willing to get good at?

I was lucky enough to ask both questions that originated at different sources, but then I do embrace randomness. It is still my time to learn. What is interesting is once you are in the workforce, earn and learn are not mutually exclusive. So go ahead make the most of it. This is an opportunity for you to find out what do you really want to do. Once you decide that, don't look back and be willing to invest the time it takes to become master of your craft.

When I am posed a question as to what is it that I do, my reply is unequivocally I solve business problems. In fact most craft's

ranging from watch making to ship-building are solving a business problem. I would define a business problem as one which upon being solved either increases profits or decreases expenses for your organization.

For example:

Writing software that predicts inventory consumption rates for spacely sprockets is a way to increase profits.

Writing software that helps reduce the number of delivery trips that big retailer has to make is a way to reduce expenses.

Building software is a lot of fun as the business problems that are solved by software have the potential to be massively scalable. Lurking in your organization is a problem waiting to be solved, maybe a process that can be automated or a missing hook to an api that returns valuable data. You are running a software company even if you aren't.

It is easy to be caught up into a title of being a Software Engineer. In reality we are solving a business problem, writing software is just the tool we use to achieve

it.

CHAPTER NINE
Finances

While the best craftsmen are never in it for the money, we owe it to ourselves to plan for our financial well being. If this is your first gig at a big corporation, then you are in luck, there are financial instruments available to you that are effectively a bonus on top of your base salary. This was confusing to me when I first started out.
So here is what you should definitely do:
1. Max out your company 401k till the matching limit. e.g. If your company matches 5% of your salary, do it. With the power of compound interest that money can grow with time. The other advantage that you get is the money that is placed in a 401k is pre-tax. This reduces your taxable income every year.
2. ESPP. Employee stock purchase programs typically purchase a company stock at x% discount for you. You need to

avail of that.

3. The most important thing to make money is to first not lose money, let that sink in. What that means is that you need to aggressively tackle your loans and stay out of debt.

It took me a good 3 years in the industry before I got on top of these things so I will save you some time. Read the MoneyMustache blog get a contrarion perspective on managing your finances. An active interest in building your stash will open options unavailable to you earlier.

CHAPTER TEN
Hobbies

To keep your sanity find a hobby outside of work. For me it is tennis. Tennis is civilized battle with the sword replaced with a racket. Tennis is about observing your opponent for that moment of hesitation. Tennis is about coming up from behind when you are down a couple of sets to win the match. Tennis is about focus and silencing your inner voice. Tennis is about living in the moment. Tennis is about believing that the next serve will be an ace, the forehand you hit will be a winner and when you barely get the ball across the net, you will stay alive to mount a counterattack. Tennis is about long term strategy and short term tactics.

Tennis is about asking questions, Can I best my opponent with a fierce forehand or a blazing serve? Or can he get the better of me in that time? Do I serve and volley or do I play from the baseline? Did that extra power on the return unsettle him? It is about adapting to your opponent and

employing the right tools. There are many tools that can be applied: the serve, the forehand, the back-hand, the slice, the lob etc. It is about asking what new tools can I employ? Tennis is about statistical thinking, what are the odds of me making this shot? Okay I admit I'm not good at this. There is this constant process of learning that goes on. Tennis is about making decisions in a split second with incomplete information.

Tennis is about showmanship and making elegant shots. I relish the crack of the ball hitting the racket on a shot and the silence of a well executed serve. Tennis is about rooting for your opponent to make that hard return because it adds to the fun. A good game of tennis is energizing. Playing tennis is about loving the battle more than winning and that is why I love tennis. Find a hobby that you love. That will make your journey as a craftsman more enjoyable.

CHAPTER ELEVEN
Health

The most important thing you can do is stay healthy. Healthy people are by definition not sick and so can work if whenever they choose to. Staying healthy is a simple task, that can be achieved with moderate exercise and the right diet. Walk a lot if you don't work out.

Distinguishing between food and entertainment. e.g. Cake, pastries, cookies are entertainment. Nuts, fruits, salads are food(thank you Scott Adams!). Once you train your brain you will find yourself making these choices automatically.

The next thing to control is the equipment you use. Software Craftsmen are susceptible to repetitive stress injuries. The way to dodge those are to:

a)Learn to type.

b) Get an ergonomic evaluation to get your posture right.

c) Get an ergonomic keyboard and a mouse or trackpad.

Investing in these now will save pain

tomorrow. I personally had wrist pain within a year of working on a cheap keyboard. That led me to explore and I discovered the Kinesis Advantage, a split keyboard that corrects the posture of your hands. I started carrying it around and boy are my wrists happy. It is also a great conversation opener when you first start work.

CHAPTER TWELVE
Reporting

Your first job will also introduce you to management. That means you will report to a superior who will assign tasks to you. Management loves to know whats going on and rightly so, they are paying you after all. There is a simple system I use. I maintain two files one called done.txt and another called runbook.txt.
I update my done file at the end of each day with you guessed it, what I've done. The runbook is more interesting. I've a short attention span. So the runbook is detailed instructions on the tasks I need to accomplish that day.
e.g.

- Write a program to parse data stream.
- Run Awesome algorithms on parsed data.
- Write methods a, b, c to accomplish that.

The advantage of doing this is that each

action item is a micro unit of work and even if I get interrupted I can resume where I left off by referring to the runbook. At the end of each week I go over done.txt collect the items for the week and send it to my manager. I've done this across multiple companies and my managers have loved it.

CHAPTER THIRTEEN
NEGOTIATION

There are many dimensions to a negotiation apart from your compensation, your stock options plan and your 401k. For me the most important one is the rate at which you can learn. You can directly leverage that to earn more. That to me is the biggest component of the negotiation. I always ask will I be happy working at this place and how much will I learn? It is important to not out negotiate yourself out of the conversation especially while you are starting out. Take the job for the adventure and growth it offers rather than just hard numbers. You can always ask for more once you have proven your worth. There are many negotiation books that you will find on amazon and you should consider reading. Negotiating is a skill worth tens of thousands of dollars each working year.

CHAPTER FOURTEEN
READ

The more you read the more models you will be able to fit in your head and identify situations conducive to your growth. Think of books as your mentors. Books are especially useful if you don't have mentors in real life. Utilize the collective intelligence available to you. You will be wiser. What books to read? I'll give you a starter list. You can use that as a starting point much like how a web crawl works. Books which I think transformed me as a person.

1. How to Fail at Almost Everything and Still Win Big: Kind of the Story of My Life by Scott Adams
2. Choose Yourself by James Altucher
3. AntiFragile by Nassim Nicholas Taleb
4. Mastery by Robert Greene
5. The Power of Now by Eckhart Tolle
6. The Surrender Experiment by Michael Singer
7. The Passionate Programmer by Martin Fowler
8. The Personal MBA by Josh Kaufman
9. The Checklist Manifesto by Atul Gawande

The other books to read are the ones by people who inspire you and you want to be like, anything that piques your curiosity.

Let that be your guiding light.

Conclusion

The ability to go to sleep knowing that you did your job with full dedication and energy is a valuable reward. There is no purpose except to enjoy the journey you are on. I wish you success and a lifetime of learning.