PRINCIPLES FOR PROGRAMMERS

THE CONDENSED ADVICE FROM THE WORLD'S BEST

BY ANDREI NEAGOIE

v1.0.0

Subscribe to get updates and new principles as this book evolves at zerotomastery.io

Preface

Advice is easy to give. That is why advice that is given should always be inspected, debated, and analyzed. What may have worked for one individual will not necessarily work for everyone or in all situations.

At the same time, as humans, we are able to learn from the past and stand on the shoulders of giants. This way, we don't make the same mistakes over and over. Instead, we get to learn from the mistakes of the past so that we start on our journey along a path that isn't completely unknown. If we are able to jump start from other's experiences, and be efficient in our learning, that is a winning strategy that will take us far.

This is the reason for this book. For the past 5 years, I have been collecting advice, articles, books and opinions of some of the best programmers, thought leaders, and business people that I know. I wanted to collect ideas from people with different opinions, and narrow down these recommendations to the core and essence. Sometimes I include principles coming from my own experience, but the goal of this book is to combine not just my own advice, but to combine the advice of the top performers and find the commonalities between them so that you don't have to start from the very beginning.

Whether you are just learning to code, or you have many years of experience in the industry, you should find something actionable and valuable in this book for your own situation.

This book is not meant to be read in one sitting. Each principle will offer its own value. Think of these as tidbits that you can read every once in a while, come back to, or be reminded of something important, so that you make sure you are on the correct path to success.

Enjoy, and let's take the leap.

- Andrei Neagoie (@andreineagoie)

PRINCIPLE 1

WHAT'S NOT GOING TO CHANGE IN THE NEXT 10 YEARS?

Build a strategy around the things that are stable in time. As a programmer, you will always have to solve problems. You will always have to effectively communicate with others, and you will always have to continue to learn and upgrade your skills. Focus a big chunk of your energy on skills that are virtually guaranteed to be valuable in the future, no matter what.

It's impossible to imagine a future 10 years from now where a customer comes up and says, "Jeff, I love Amazon; I just wish the prices were a little higher." "I love Amazon; I just wish you'd deliver a little more slowly." Impossible. When you have something that you know is true, even over the long term, you can afford to put a lot of energy into it.

— JEFF BEZOS

PRINCIPLE 2

Helping others without answers

Programmers don't grow by being given the answers. They grow by trying things, experimenting and solving problems themselves. These are the skills and lessons that will serve them well as they level up their career. Instead of giving them the answers, give them the tools to find the answers. Help others by letting them reach a conclusion or solution on their own. Just guide them in getting to that goal without holding their hand.

Beware of the Half-Life

Everything we know, not just about programming, has an expiration. The half-life of knowledge is a term to denote the time it takes for half the knowledge in a domain to be outdated. According to one study, the half life of an engineer in 1930 was 35 years vs 10 years in 1960[1]. Imagine what it is now. Because technology is moving so fast, eventually the ability to solve problems, and continuously learn, will become the most sought after skill. This is a muscle that you can train.

1. https://spectrum.ieee.org/riskfactor/computing/it/an-engineering-career-only-a-young-persons-game

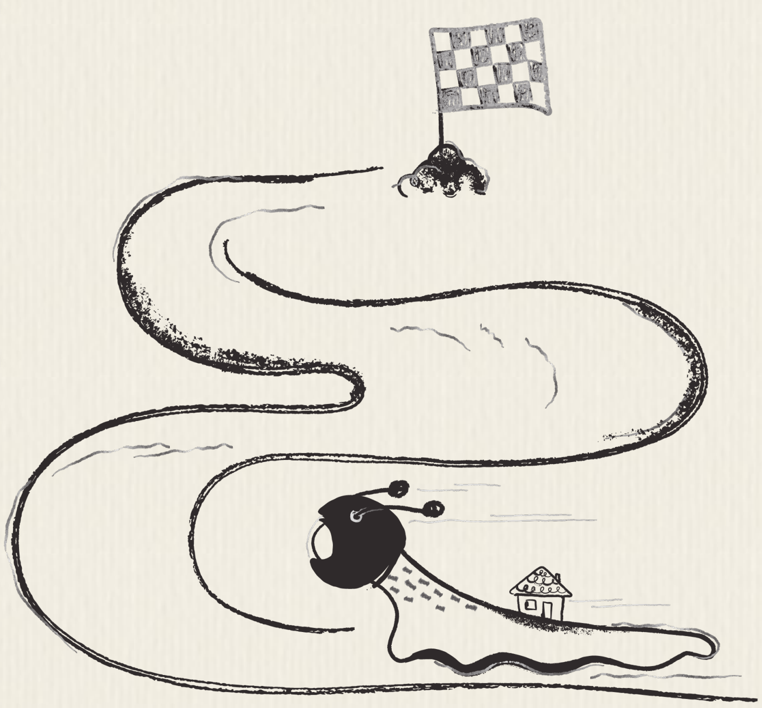
Experience is not created equal

10 years coding by yourself with no other programmers around you, with no code reviews, doesn't necessarily make you better than someone who has 1 year of coding experience working on a project with team members and getting code reviews daily. Get out of your bubble. Work with others, be surrounded by other programmers and learn from them. The biggest thing you can do if you are starting out is to get hired as soon as possible so you can work with others to accelerate your learning. Let others see your work. Get feedback. That's how you gain experience.

PRINCIPLE 5

DON'T FOLLOW THE MONEY, FOLLOW THE BRAINS

Great pay and salary today is good. But if you want to succeed long term, you are better off taking a lower pay to be surrounded by smart people and being in a situation that allows you to learn from others. If you can get both, that is even better, but always have this as a priority when picking between options: Which option will allow me to learn and grow the most and be surrounded by people smarter than me?



PRINCIPLE 6

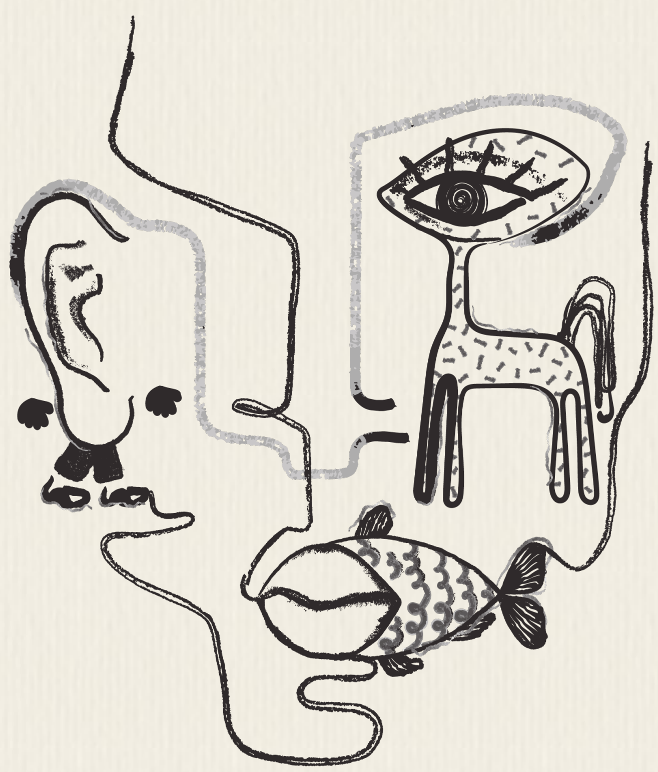
THIS IS NOT NEW

Most problems you will encounter in your career as a programmer have already been solved. Unless you are doing academic research, or you are really pushing the boundaries of a fast growing field, most of the time you are simply doing something that has been done in the past. This is not to diminish your work. Instead, it is a reminder that using proven tools and techniques are often a better approach to solving your problems instead of reinventing the wheel.

PRINCIPLE 7

Treat Non Technical People with Respect

Be respectful to those who are not technical. Everyone has their own strengths. Instead of judging, try to help, try to communicate, and try to learn from them. Patience and respect goes a long way. This is an opportunity for you to practice your communication skills to deliver your message.



PRINCIPLE 8

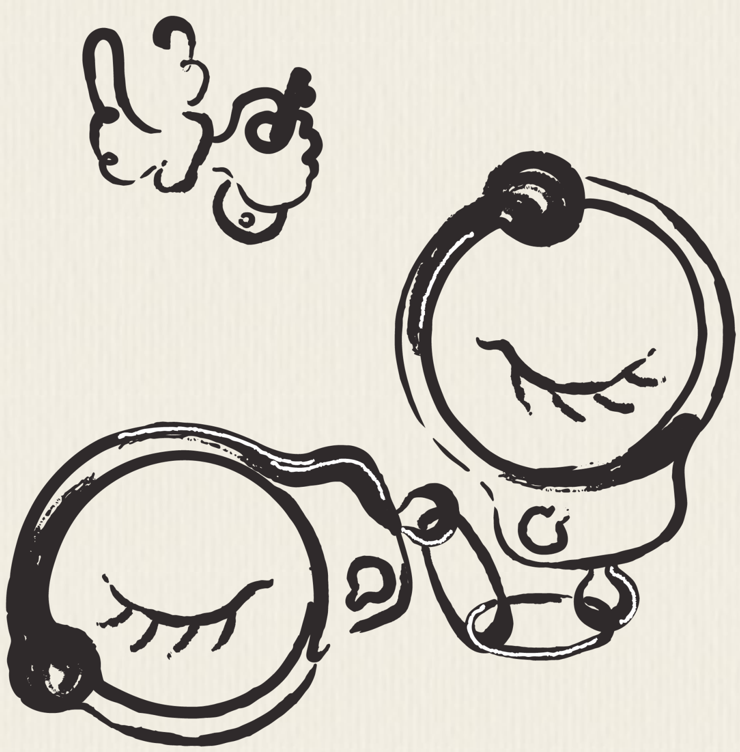
There will always be people smarter than you

Accept this fact. You don't have to be the best at everything. Learn from masters and combine different skills into a unique way to offer value that only you can provide. Accept feedback, accept advice, and accept criticism. Ego is the enemy. Always be open to new and different ways of doing the things you think you are great at.

PRINCIPLE 9

This too shall pass

Whenever you are struggling, stressing, or finding something difficult, realize that it is all temporary. Are you often getting down on yourself? Thinking you aren't good enough of a programmer? Anything worthwhile in life will push your boundaries. If it was easy, everyone would be doing it. If you get down, or you think you can't do this, change your mindset. No matter what hurdles you are trying to get past, this too shall pass.



Use The Feynman Technique

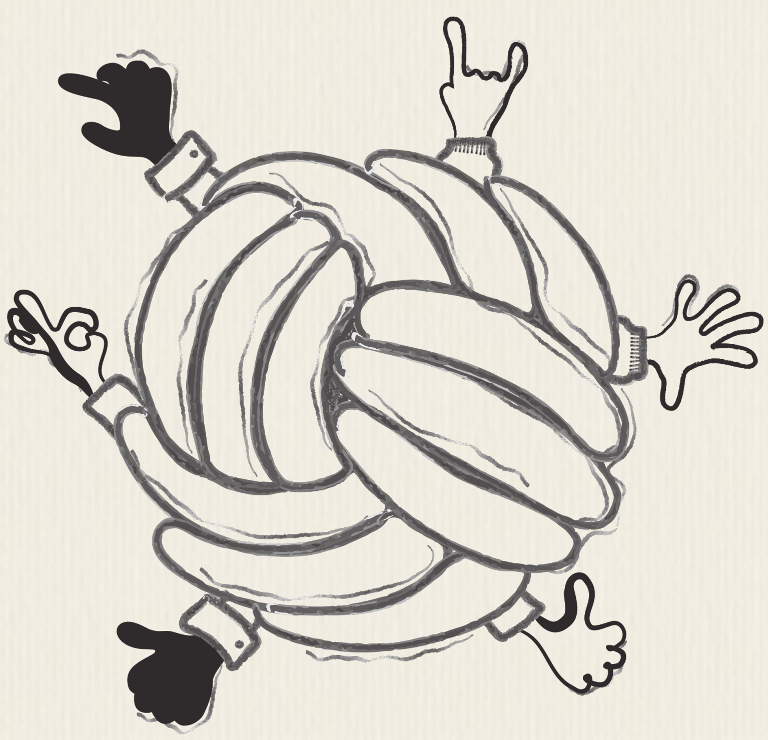
In order to truly understand something, you should be able to explain it and communicate it easily to someone with no prior knowledge of the topic at hand. If you are able to speak to your colleagues without jargon as if you are talking to someone with no technical knowledge, it frees us from hiding behind big words that we may not fully understand. Ask yourself questions like “how does this work?”, “why is this done this way?” If you can’t answer that, research and learn until you can explain the topic to anyone.

If you can't explain something in simple terms, you don't understand it.

— RICHARD FEYNMAN

Divide and Conquer

When solving problems, don't try to solve everything at once. Break it into smaller problems and solve each one at a time. By solving these sub problems, you then connect the dots. Every big problem will be intimidating, but by breaking it down, it becomes more manageable. This applies in corporate structure, developer teams, algorithms and many more. Start as simple as possible, then connect the dots to reveal that tough problem you initially thought impossible.



Controlling complexity is the essence of computer programming.

— BRIAN KERNIGHAN

PRINCIPLE 12

It is always easier to add than to subtract

Writing code is easy. Removing it is hard. Adding a new feature to a product is easy. Removing it might break the system, or may result in angry customers. Addition leads to more chances for errors and mismanagement. As a programmer, we are paid to improve the value of the business. Every time you write code or introduce third-party services, you are introducing the possibility of failure into your system. The number of lines of code you write does not make you a better programmer. Your job isn't to write code. You are not paid to code. It's to add value[2]. Many times, less is actually more.

2. https://bravenewgeek.com/you-are-not-paid-to-write-code/

Measuring programming progress by lines of code is like measuring aircraft building progress by weight.

— BILL GATES

Start with Why

The ability to explain why you chose this particular solution or why you chose this tool over alternatives is a sign of a great programmer. Don't just tell people. Explain yourself and give evidence for your thinking. Don't do things because you read it in a blog post. Be able to explain the "why" of what you are doing. If you are able to explain with evidence, you are ahead of most.

PRINCIPLE 14

Notice technical debt

The bigger an application's codebase, the more bugs are introduced whenever a new feature is built. Eventually, the rate of work created from new bugs cancels out the rate of work done from new feature development. This is known as "technical debt" and is the main challenge in professional software development[3]. Learn to diagnose the beginning of technical debt.

3. https://www.csc.gov.sg/articles/how-to-build-good-software

There are only two hard things in Computer Science: cache invalidation and naming things.

— PHIL KARLTON

PRINCIPLE 15

Compound Interest of Learning

We overestimate what we can do in the short term. We underestimate what we can do in the long term. Learn the concept of compound interest. This doesn't apply just to money. It applies to knowledge. When we learn something new we feel like we will never get there. You see someone you admire and think that you will never be able to get there. Of course you won't…in the short term. However, if you are able to learn each day just a little bit, eventually you will get there. The key is to keep improving every day, and keep learning. Years from now you will look back and realize how far you have come. We miss deadlines all the time because of short term optimism, but long term, we can accomplish a lot more than we think. A little bit of learning every day will always trump trying to learn a lot in a short period.

THE 2 TEACHERS

Find the right type of mentor by identifying the 2 types of teaching:

1. Teaching students and holding their hand so they will always come to you for answers.

2. Teaching students so they outgrow you and are able to find answers on their own.

#1 is a salesperson. #2 is a real teacher.

PRINCIPLE 17

Pick the Right Tool for the Job

A good programmer knows to pick what tool to use and when. This doesn't mean you have to learn every single tool under the sun. You need to understand what tools exist and the pros and cons of each so you can make a smart decision of what to pick when the time arises. Remember this: Find the right tool or system to use. Learn how to customize them to fit your unique requirements and fix problems discovered along the way.

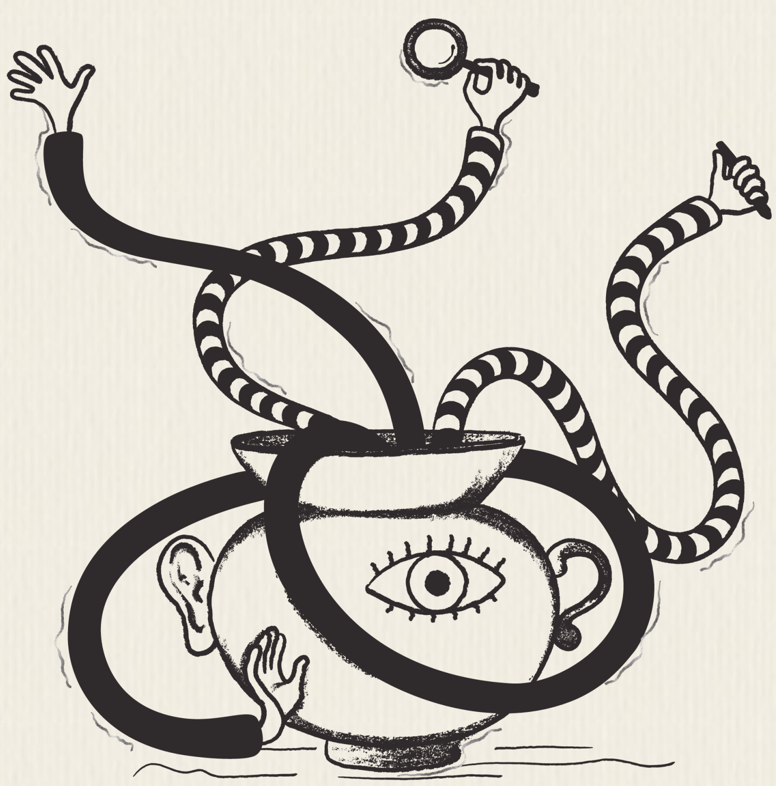
PRINCIPLE 18

Don't Write Code Right Away

You need to think before you code. Code after all, is not only a way for us to communicate with a machine, but also to our team members. It's easy to jump straight into coding and it may work on smaller projects, but the larger the project gets, the more you have to think, plan, and strategize what you should write. What you want to do and how you will do it should be clear to you before you start typing. At the same time, don't over plan. Sometimes "good enough" is better than over engineering. As with most things in life, there is no black and white. Just a nice balance. Don't over-plan but also don't write code right away.

Don't copy paste code

If you are using code you don't understand, stop. Understand it first before you add it into your project[4]. Learn from the solutions of others, but you should understand what you add to your projects.

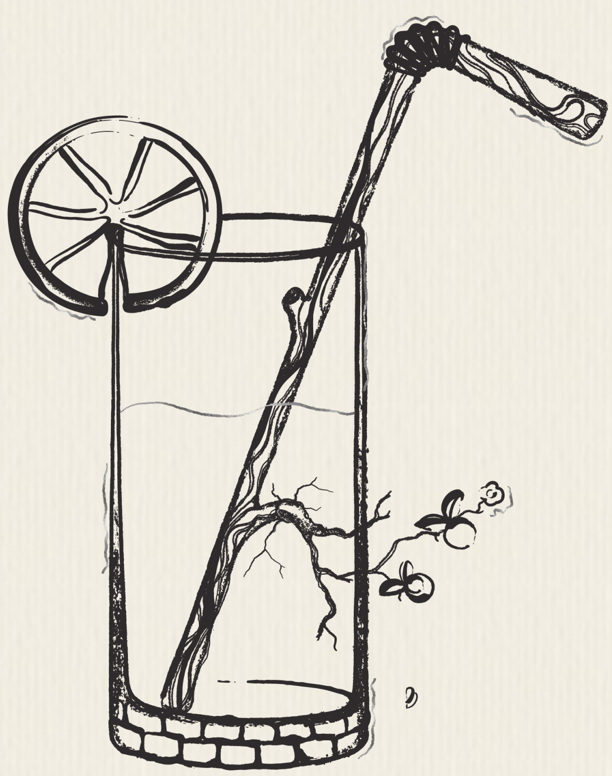


4. https://daedtech.com/5-things-ive-learned-in-20-years-of-programming/

PRINCIPLE 20

Working code is not always right

Just because the code is working, it doesn't mean it's right. Working code is great, but does not necessarily represent the optimal solution. Working code is just one step in an iterative process of continuously improving and adapting our code to the situation. This is why the first solution to your problem is not necessarily always the right one.



Simplicity is the ultimate sophistication.

— LEONARDO DA VINCI

Fix the problems that matter

A good programmer fixes the right problems, not all problems. There is limited time and limited resources. The good programmer is able to prioritize and focus on what matters most.

The art of debugging is figuring out what you really told your program to do rather than what you thought you told it to do.

— ANDREW SINGER

PRINCIPLE 22

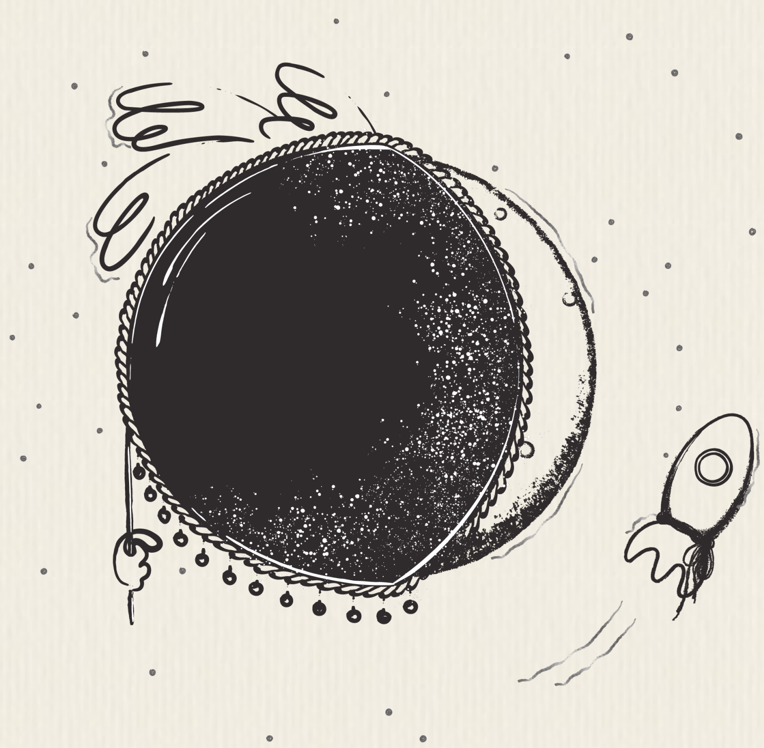
THE SECRET TO EMPATHY

Focus on your own weaknesses, not others' weaknesses. You don't gain empathy by talking. You gain empathy by listening. Value your relationships with other programmers as how you interact with them is part of what makes you a programmer. If you are not a good person to work with, you aren't reaching your full potential. Always remember that no matter how difficult someone may be to work with, everybody is fighting their own battles that we may know nothing about.

PRINCIPLE 23

Learn to use Data Structures

One of the most foundational skills you can learn as a programmer is the available data structures in your programming language of choice and when to use what. Learn about time complexity, space complexity and Big O notation. It will immediately make you a better programmer. Learn the difference between Arrays, Hash Tables, Trees, Linked Lists, Queues, Stacks and Graphs as a baseline. It will take you far in your career.



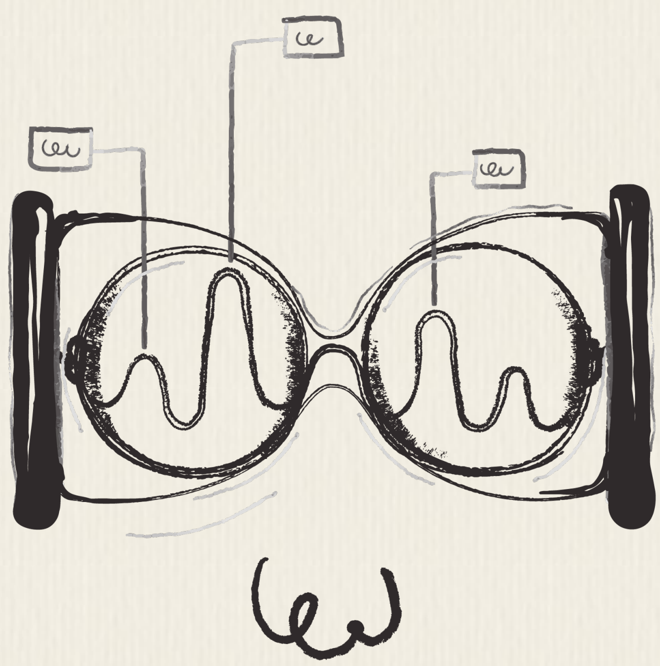
Premature optimization is the root of all evil.

— DONALD KNUTH

PRINCIPLE 24

There is no Yoda

No single person knows everything. Even the people you respect, your seniors, and the tech leads all have flaws. They all have gaps in their knowledge. Don't assume everything they do is always right. Be able to question it and be willing to ask questions when you don't understand something or don't agree with something. Don't assume they are always right and don't follow blindly.



Don't obsess about best practices

Best practices are not always going to last. Good practices with our current level of understanding and tools is a better way to look at things. What may be good now may not be the best in the future. Be open to change and be open to different opinions and thoughts. Sometimes the ideas that cause the most controversy today will turn into the best practices one day, and those in turn will one day become outdated.

I was too busy looking at the trees in front of me to see the forest all around.

— FAMOUS PROVERB

PRINCIPLE 26

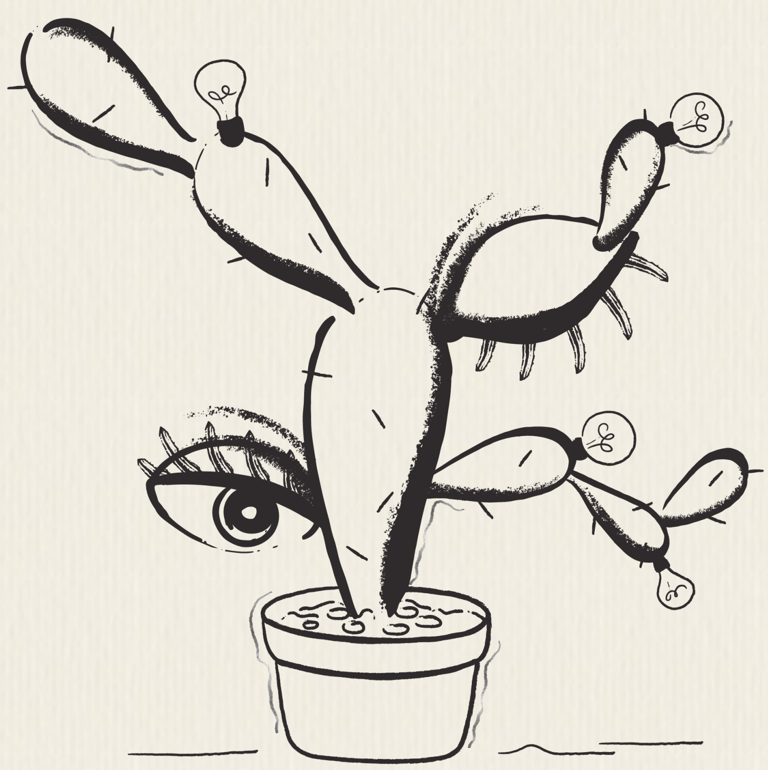
YOU ARE NOT YOUR CODE

Don’t let code reviews hurt your feelings. That one comment that was made on your code being bad will very quickly disappear from your memory. Criticisms are fleeting. Every code review is an opportunity for you to learn. An opportunity for you to not make the same mistake again. No matter who you are there are always going to be things that you wish you could change, you could have changed, or you will change in code. Detach yourself from your code. Those who understand that will outpace those who take code reviews as an attack on their ability. Iterate and get feedback as quickly as possible. That’s how you make good software.

PRINCIPLE 27

Learn to Say No

This applies at work, in learning, and in your career. You have limited time with unlimited things to learn and do in your career. Learn to say “NO” to the things that aren’t critical. Learn to prioritize and identify the core things you need to focus on. Determination and hustle without efficiency leads to burnout.



PRINCIPLE 28

ASK QUESTIONS

Be honest. If you don't understand something, ask. This should be instinctive. No one is keeping score and in the long run you are going to learn more than others who stay silent. Ask questions. Make it your nature.

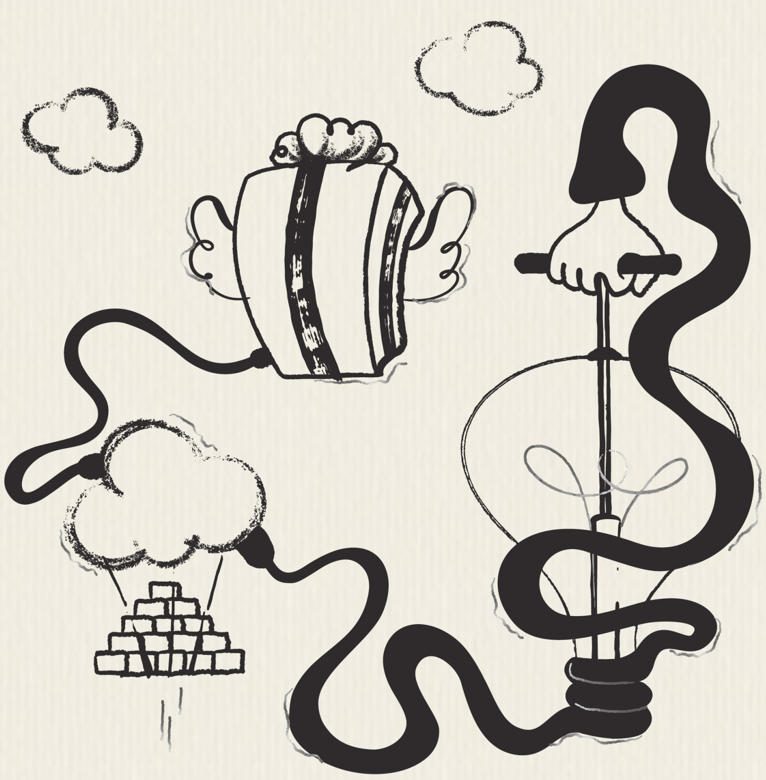
PRINCIPLE 29

There is one constant in this world

There is no right and wrong. Only pros and cons. Every situation, every library, every opinion, every line of code, offers a pro and con to a specific subject. The less you see things as black and white, the more you are able to think in terms of a balance and tradeoffs.

There is no perfect

All products, all projects, all codebases have bugs, complexities and issues. You can always make things better. There is no perfect tech stack. Don’t be that programmer that complains about how poorly somebody has written this piece of code. You don’t know the context and the time limits the person had. Just understand that nothing is perfect. Everything can be improved, or made worse. Accept it.



Perfect is the enemy of good.

— VOLTAIRE

PRINCIPLE 31

Depth, then Breadth

Narrow your focus at the beginning of your career. Figure out the chosen path to getting hired and focus deeply on the subject that you know you will get hired with. From there, once you feel you truly know the subject, expand to other languages, fields or topics. Trying to do too many things at once and learning everything at once is a recipe for disaster. You can only do one thing well at a time. Do that, then look beyond.

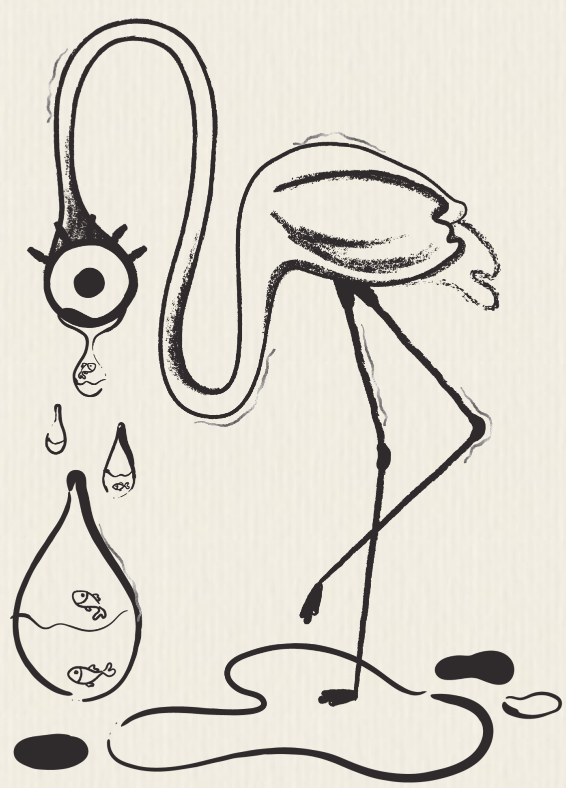
Know-it-alls are never all knowing

Don't be a know it all. If you have real world experience, and you are speaking from experience, then do so. Provide context to your opinion, but don't just say things because you read it in a blog post or you heard someone say it online. Form your own opinions, and be open to others' opinions. No matter how much experience you have, you will always have gaps in your knowledge that can be filled by others.

PRINCIPLE 33

Help Others with Code

Software is used to help people. How does your code help someone? That is the guiding principle. How does it help your colleagues, your boss, your partner, your users? The main value in software is not the code produced, but the knowledge accumulated by the people who produced it[5].



5. https://www.csc.gov.sg/articles/how-to-build-good-software

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

— JOHN WOODS

PRINCIPLE 34

YOU AIN'T GONNA NEED IT[6]

Always implement things when you actually need them, never when you foresee that you need them. If it is a "nice to have" then you probably don't actually need it.

6. https://ronjeffries.com/xprog/articles/practices/pracnotneed/

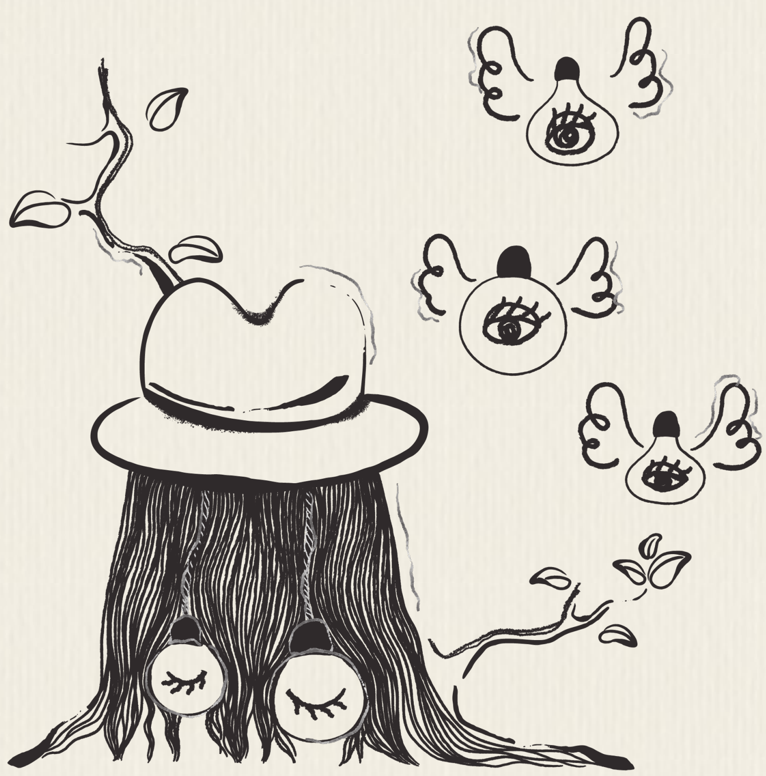
One of my most productive days was throwing away 1000 lines of code.

— KEN THOMPSON

PRINCIPLE 35

Don't Complain

No matter how menial the task, how frustrating the client is, or how disorganized you think a process is, complaining doesn't accomplish anything. You are hired to solve problems, not to complain. It's easy to complain to sound smart and appear you are above it all. It's a lot harder to work with what you are given and do your best work. If you are going to discuss an issue, always have at least one solution to propose. You're not always going to work on the most interesting tasks but you can shine by working within the constraints that are given to you.



PRINCIPLE 36

LET PEOPLE SPEAK

The hardest part of working on a team is giving everyone permission to express ideas and opinions. If you are able to create that environment, you will stand out. Don't let people self-censor to avoid looking incompetent. Give people the permission to speak their minds. Congratulations, you are now better than 90% of managers out there.

PRINCIPLE 37

PAY FOR YOUR EDUCATION

There are a lot of free resources online on any topic you want. But the fact of the matter is, paid forms of education is usually higher quality. Most importantly, when you pay for something, you are more incentivized to follow through with finishing it. Education is the best investment you can make. Save money by not buying a cup of coffee, but don't skimp out on a good education.

PRINCIPLE 38

Be Willing to Change

Your favourite tool, language or library probably will be different in 10 years. Be willing to change your opinions, tools and beliefs. Be willing to say that things that you know now may be different or opposing in the future. This includes updating your skills and being aware of new tools that come up that are better and more efficient. Don’t be stuck in your ways. Tools are always improving, and so should you if you want to stay relevant.

You can't predict, you can prepare.

— HOWARD MARKS

PRINCIPLE 39

Beware of Attribution Bias

This is a fundamental cognitive bias that humans have. We attribute bad behaviours of others as caused by something internal (i.e. who they are or their personality) while we attribute bad behaviours of ourselves as caused by external factors (i.e. I didn't get enough sleep, or it was somebody else's fault). Don't fall into this trap.

PRINCIPLE 40

There are no easy routes

At the end of the day, if you want to be exceptional at a skill, there are no shortcuts. You need to work hard and you need to push your boundaries where you are struggling to solve problems. Hard things are worth it in the long run. Watching Youtube videos on programming is easy. Building your own project from scratch and launching it is hard. If you feel uncomfortable learning a new topic, then good. You are doing something worthwhile to improve yourself. If you're not struggling, you're not trying.

Quit when you'll be mediocre, when the returns aren't worth the investment, when you no longer think you'll enjoy the ends. Stick when the dip is the obstacle that creates scarcity, when you're simply bridging the gap between beginner's luck and mastery.

— RYAN HOLIDAY

Final Words of Advice

This whole journey is a marathon and not a sprint. If you enjoy every single day and the work that you are doing, you are more likely to flourish and keep doing what you are doing 10 years from now. The best thing you can do for your career is to make sure that you enjoy this learning process, you enjoy your work environment, and you do everything in your power to maintain that happiness.

If there is one thing you should have taken away from this book, it is that being a great programmer requires more than just technical knowledge. Learn to communicate. Be someone that people love working with. The more other people want to work with you, is a direct indication of how successful you will be in your career as a programmer.

If you need to take breaks, take breaks. If you need to quit because of a toxic boss or manager, then leave. Do not compromise on your happiness. Those who enjoy their journey are those who will be rewarded in the future because they continue on their path to mastery.

Spend time away from programming. Enjoy your friends, family and life. Programming isn't everything. You don't need to be sprinting and learning every single moment of your life. Enjoy the process and enjoy a good balance.

Go make good decisions, and good luck!

Find out more about my work and the classes I teach at:

[WWW.ZEROTOMASTERY.IO](http://www.zerotomastery.io)