10 Signs You Will Suck at Programming

Jonathan Bluks Follow
Jan 27 · 13 min read

More Stickers Doesn't Make You More Better.— Photo by Tim Gouw on Unsplash

I often see questions on Reddit or Quora along the lines of "How do I know if I'll succeed as a programmer?" *(In fact this post expands on an answer I gave on Quora a while back.)* When someone is considering a career change, or is interested in software development and curious about what it takes, inevitably the question of programming comes up.

In fact, I think this is a major barrier to entry in people's minds when they don't have any formal training in computing. It's natural to think that if you aren't good at programming, then your desire is a non-starter. It's kind of like if you want to be an actor and wonder if you'll be good at acting.

As an Educator that teaches Full-Stack Web Development, I have taught many "first time programmers". And the good news is that I have rarely found a student that couldn't learn to program. I see it as a basic human skill, just like reading, writing, and arithmetic. Anyone can do it, it is part of our human capacities, but does need to be learned.

Over the past two years of teaching, I have witnessed various students struggle in the process, and some common themes that come up in their struggles. *If you look at this list and see them in yourself, rest assured, you will truly suck at programming* and should probably find something else to do with your time. But, if you are still committed to your goal of becoming a developer, you can easily face these issues and change.

> *Programming is a basic human skill, just like reading, writing and arithmetic. Anyone can learn to program with time and effort.*

. . .

The following list will help you know if you will suck at programming —and what you can do about it if you want to change.

1 | Lack of curiosity

> *If you lack curiosity about computers and how technology works, you will never be successful as a programmer.*

A fundamental requirement for learning is an active interest in the thing you are learning. If you do not possess a mind that is curious about technology, you will not have the energy it takes to persist in learning the broad and deep knowledge required to be a successful programmer.

In contrast, the world of technology is like a huge ocean of interesting domains, inter-connected ideas, and possibilities that can excite the imagination. It takes an inherent internal motivation to want to dive in and discover all that you can.

> ***Find Your Curiosity:*** *Ask yourself if programming truly interests you. If your honest answer is that it doesn't, go find something that you are interested in. Save yourself the time and energy. But if your answer is "Yes", then push yourself find something new that you haven't noticed before, recognize the vast ocean and dive a little deeper.*

2 | Lack of autonomy and resourcefulness

> *If you don't develop the ability to solve problems for yourself, you will never be successful as a programmer.*

Without a doubt, to be a successful developer, you have to be confident in your OWN ability to learn. This is actually a fundamental life skill—if you are are over the age of 18, nobody is obligated to teach you anything. That's reality. It's up to you to find the information and help that you need to learn what is important to you.

In the world of development, all the information you need is found in that magical place formerly known as the *Information Super Highway*. This massive library has one huge doorway: Google. Learning that you can simply type whatever you want into Google and get to the information you need is the first hurdle to cross when you want to learn the skills needed to be in technology.

In addition to being a good googler, all programming languages have documentation and specifications that are **very** explicit about how the language works. It's like using a dictionary—when you see a word you don't recognize, you look it up. The quickest, most reliable way to build your skill as a programmer is to simply read the documentation. It's literally all there.

> ***Use the Resources:*** *Recognize that all the answers you need really are out there. When you have a question, force yourself to google it before asking someone else for the answer and check the documentation. Save the time of others for when you have tried and truly failed to find the answer you need.*

3 | Lack of persistence in the face of a problem

> *If you give up easily in the face of problems, you will never be successful as a programmer.*

The essence of programming is solving problems. That's the whole reason computers were invented! Whenever you begin working on a program you will encounter a whole "stack" of problems. And once you resolve a problem, there is almost always another problem right behind it. You *are* making progress, but there are always *new* problems to face.

Facing that stack of problems can be daunting and discouraging. If you feel like things should "just work", then you won't have the energy to persist as the problems continue and little by little knock down your emotional resolve. It's literally your job to figure out why things aren't working.

From my in-class experience, there are usually one or two students per class who seem to have knack for encountering more problems than other students—often quite random and obscure problems. I remind the student that the more problems they face upfront, the possibility of learning more deeply and thoroughly increases. If they can gain understanding through these problems, they will quickly find that they are more confident because they have faced and resolved more problems than the average student.

> ***Patient Acceptance:*** *You need to recognize that problems come with the territory and are not problems, but in fact challenges. Every challenge you face and overcome gives you deeper understanding and a better ability to face new challenges, and quickly resolve old ones.*

4 | No feeling of success in overcoming a problem

> *If you don't feel a sense of excitement and accomplishment after solving a problem, you will never be successful as a programmer.*

Related to the previous issue of giving up too easily is a lack of "good feelings" once you have successfully resolved a problem. When fixing bugs and issues becomes a treadmill that never seems to end, you lose touch with the excitement that comes in overcoming a problem.

There is actually a dopamine hit that you need when you overcome a problem. This is similar to the experience of completing a level in a video game, or solving a challenge like a crossword or sudoku. We all know that there is a good feeling that comes from persisting through a challenge and then finally winning at the end. But if you have lost the ability to feel those feelings, or never truly cared in the first place, you will not be able experience the joy that comes from programming. If you see programming as a grind where you just want to get a result as easily as you can, you will never truly be a successful programmer.

> ***Celebrate Your Wins:*** *Whenever you solve a problem that you struggled with, no matter how small, always take pride in your accomplishment, take a break and congratulate yourself for a job well done. Let the feeling of success sink in and energize you for the next problem you face.*

5 | Impatient about learning and understanding

> *If you are impatient about learning and expect to master everything quickly and effortlessly, you will never truly be successful at programming.*

As humans we are limited creatures. Even though our world moves faster and faster, and computers are a big cause of that, we can only move as fast as we are able. Our brains work at a certain speed, and depending on our past, our beliefs, our emotional states, our health… we will all learn and integrate information at different speeds.

The world of technology is like a vast ocean. You will never get to the end of it, you will never get to a point where you are a master with nothing else to learn. If you let yourself get overwhelmed, you will always feel a pressure to "catch up" and feel that you never know enough. If you can't accept what you know and then learn a little bit more, you will feel like you are getting nowhere, and give up.

Instead, you need to enjoy the journey of learning for itself. Every little bit of knowledge you gain, or new skill you gain needs to make you excited. Like solving problems, you need to let yourself feel the pride of recognizing that you have taken a step forward, even if it is a small one.

> ***Acknowledge Your Progress:*** *There is a lot to learn, and the journey of programming never ends. But the knowledge is cumulative, so take pride it what you do know, and trust that every effort you take in your learning will create a strong foundation of knowledge for wherever your career takes you.*

6 | Getting bored/tired from thinking

> *If you are lazy in your thinking and see concentrated, focused thinking as a chore, you will never truly be successful at programming.*

Programming is a thinking activity. As humans we are really good at thinking, but the reality is that even though we naturally spend all day doing it, we are lazy in our thinking. The ability to maintain concentrated, focused effort on a single problem over a period time is difficult if you aren't used to it.

Symptoms of this include staring blankly at the screen, feeling a cloud descend on your thoughts, procrastinating on a problem, flipping between browser tabs, and desperately scanning StackOverflow for "an answer". These are signs that you have hit a mental limitation and need to find a way through.

While programming, you will get tired and thinking literally burns physical energy just like exercising our bodies. When you aren't used to exerting the mental energy needed to it can feel difficult to stay focused. But it is just like going to the gym, the more that you do it, the stronger you will get.

> ***Your Mind is a Muscle:*** *Trust that your brain is like a muscle—as you use it, it gets better and more efficient in it's thinking. As you put pieces together and develop mental concepts, it becomes much easier to discover solutions.*

7 | Inability to think for yourself

If you expect others to think for you, and are unwilling to look at the details of your own situation, you will never be a truly successful programmer.

When you are learning something new, it is easy to feel like you lack the knowledge and experience to have your own opinions. Taking initiative or doing/saying the wrong thing seems risky.

There is an inherent fear we all have in being wrong. When that fear of being wrong inhibits your exploration and curiosity, you stifle the ability to develop true knowledge, knowledge that is gained from experience and "failure". When you need to rely on the "guru's" opinion, a popular blogger, a best practice, or the "textbook" answer, then you have not truly integrated a working knowledge of programming.

You need to develop you own opinions about what works and what doesn't. You need to understand why you think your solution works, and what the benefits are. You need to developed a nuanced perspective that goes beyond what is obvious. You need to be able to "argue" your side, and then if you change, you can own the new perspective you have gained.

> ***Think for yourself:*** *Through your experience and critical thinking skills, develop your own perspectives. Make reasonable guesses, take a position, and be willing to change as new information comes to light.*

8 | Rigid, narrow and/or disorganized thinking

> *If you are rigid in your thinking, and you have difficulty keeping your code organized — and by extension your thoughts organized and focused, you will never be successful at programming.*

There are two extremes that I sometimes see in students. The first is a rigid and narrow approach to thinking. This attitude refuses help, and despite feedback, doesn't change. Things are seen from one perspective only, and suggestions are ignored.

A second extreme I see is disorganized thinking. Students seem to make things more complicated than necessary, their code is messy and hard to follow. They overthink problems and write 100 lines of code, when 10 would have been sufficient.

When these two mindsets are combined the result is an uptight approach to programming, like a brute force approach that leads to layers and layers of fixes and "hacks". What is required is the ability to go back over the solution, reevaluate it, give up on the initial approach, and reorganize.

Not being able to see other possibilities or receive feedback inhibits the ability to grow and improve. Being disorganized slows you down and prevents you from seeing patterns that would otherwise be obvious. And the overall quality of your work is diminished.

> ***Self Reflection:*** *You need to take a step back, and look at the bigger picture of how you are approaching things. How can you do this better? Is there anything you can do to make your life easier? What are you missing that could help you?*

9 | Needing the "right" answer instead of recognizing a spectrum of "good" and "bad" answers

> *If you see the end goal of programming as finding the right solution, rather than a spectrum of solutions, you will never truly be successful as a programmer.*

When starting to learn the skills or programming, students often want to know whether what they have done is "right". The answer is always "it depends."

Computer Science is a science of evaluating tradeoffs. Given different sets of circumstances, which would be the better path? It all depends on the circumstances and goals. When you see programming as a test with either a right answer or wrong answer, you are losing sight of the bigger picture, and giving up your creativity. Any answer can be "right" if you can justify it given the circumstances.

The reality is that programming is more like writing poems or short stories (or potentially novels if the programs are large). There is an aesthetic and beauty that can be seen in your code, and sometimes it is only recognizable by you and other programmers. The reasons you decided on your solution, and the way you conceived of your answer is more important than the "right way" or the "wrong way". Having the mind of an artist allows you to play with options and possibilities, rather than thinking there is only one way. That is the beauty of programming, there are many ways to solve a problem, and the consideration of different possibilities leads to a feeling about which way is best for the situation.

> ***Get Creative:*** *Recognize that there are numerous ways to solve a problem, and through experience and exposure, you will develop a nuanced understanding overtime about which solutions feel better than others. Looking at the big picture, imagining different possibilities and trusting your gut will lead to better solutions that are more satisfying.*

10 | Not paying careful attention to details

> *If you gloss over details, and overlook little things, you will never be a truly successful programmer.*

Computers are precise machines. When it comes to programming a computer, you need to provide the necessary commands *precisely* in the way a computer expects. If you don't, nothing will work. There isn't a middle ground where things mostly work—it either does or it doesn't.

That means that when you are programming you have to have an eye for details. Every space, bracket, or semi-colon counts. When out of place, nothing will work. When the computer spits back an error message, you have to be able to look at that message and understand exactly what it is telling you. And the reality is that if you miss details like that, you could spend hours tracking down a problem that was literally the result of a typo.

As they say, the devil is in the details. And that is definitely true for programming.

> ***Pay Attention to the Details:*** *The details matter and you have to accept that. Once you do, you can begin to always be scanning your code for anything that is out of place. You can organize your code and use tools that help you identify issues quickly.*

Bonus: Being Business-Minded

This is a side observation that I have noticed—students who are particularly business-minded, are often focused on the outcome rather than process. They want to get a "working app" that will move them forward on their business idea, they want to "get to market first", and they see the learning curve as a barrier to their goal of getting their business going.

In reflecting on students that were difficult to truly help grow as programmers, I found an impatience with the process inhibiting the learning needed to truly understand technology. They tend to see technology as a means to an end, rather than a legitimate domain of knowledge to be explored and enjoyed.

As a natural extension of this, I have found certain students that were more business inclined, and struggled with their learning, often quickly had freelance clients signed up for work that they didn't actually have the skills do themselves! They would scramble to find resources/templates to get a project working for a client, or, outsource the work to someone else. *They truly sucked at programming, but were amazing at getting people to pay them to program!*

So what I would add, is that students who desire to start businesses, are excellent at sales, networking, and business development may struggle more than others to learn the skill of programming. Their natural desire to create financial opportunities and connect people to solutions makes them impatient with the tedious details involved in programming.

. . .

Conclusion

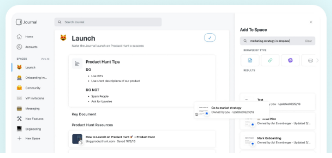
While programming can be a difficult skill to learn, it is certainly one that most people can learn. The above list contains attitudes and mindsets that get in the way, but most people can overcome them and develop a competency in the area of programming—if not mastery.

If you are interested in learning to program, I encourage you to begin the journey. Keep the above list in mind, and start exploring the many resources available online that can get you moving forward quickly. You won't regret it.

. . .

Read next: What would be possible if all our thoughts were connected and easily accessible?

Meet Journal →



📄 Read this story later in Journal.

🗞 Wake up every Sunday morning to the week's most noteworthy Tech stories, opinions, and news waiting in your inbox: Get the noteworthy newsletter >