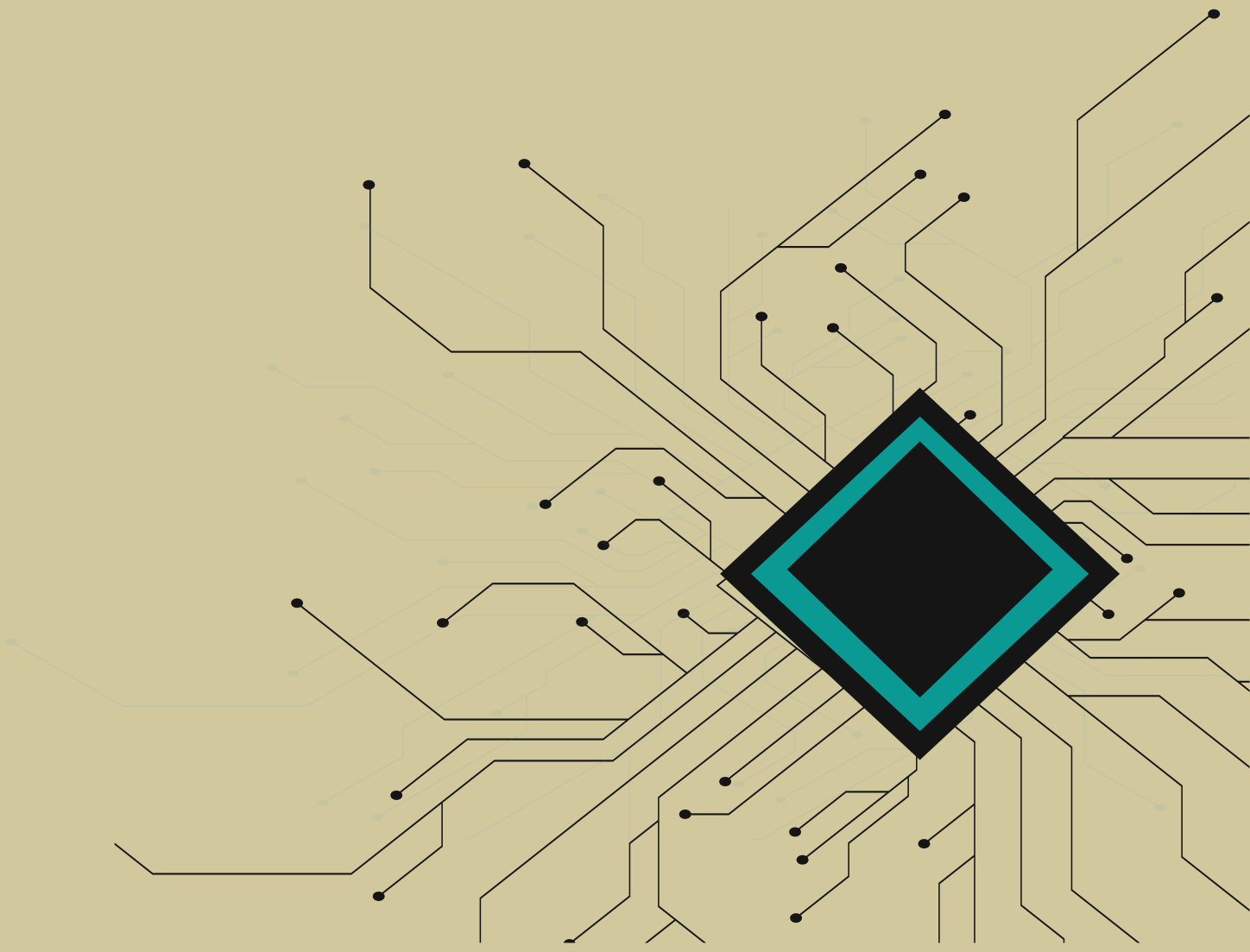
SELF-TAUGHT PROGRAMMER

# STUDY MANUAL



WWW.ANDYSTERKOWITZ.COM

# PURPOSE

I've had the honor of being able to share ideas about software development and learning how to "teach yourself" to code  on my YouTube channel for over 2 years now.

Over the course of that time I have seen that often many people's issues revolve around the inability to learn complex skills. A complex skill like programming or learning a new language isn't something that can be learned by simply reading through a textbook or watching a few how-to videos.

The attainment of complex skills require long periods of study, repetitive practice, a great deal of frustration and very slow progress.  Those that can persevere through this process can reach the other side with a skillset that can provide a lucrative career and a new level of self-satisfaction.

The point is: there is no way around this difficult process of building a skillset that is complex in nature.

In this manual I have put together the most useful advice for studying effectively that I have discovered in my years of being both a software developer and a coach to aspiring programmers.

The points contained in this manual are principles that, when applied, amplify the results you will get out of the time you spend studying.

So please make sure to apply these points to your own practice and if they work then make sure to share them with others.

Best of luck,

Andy Sterkowitz

# SELF-MANAGEMENT

*"Sow a thought and you reap an action; sow an act and you reap a habit; sow a habit and you reap a character; sow a character and you reap a destiny."*

You don't become a programmer overnight. It takes repeated practice over and over spread out over months and even years to have a strong grasp of how to develop software.

A majority of those looking to become self-taught programmers will focus a lot on the technical skills and often ignore their inability to create and stick with habits. They often think their problem is that they need to understand the syntax of Javascript better or need a better explanation of Python's class library.

From my experience however, I have found that most people lack the ability to manage themselves which often times would resolve most of the common problems. Personal management is **the art of getting yourself to do what needs to be done in order to reach your highest level goals**.

For example, you need personal management if you want to get into excellent shape. You have to carefully watch what you eat. You have to plan your day to figure out when you will exercise. You have to come up with creative strategies to avoid eating unhealthy foods. You have to be flexible enough to change your schedule as things come up.

Without these skills many people will fail. They'll blame their willpower or their genetics but they'll never think twice that they have effectively failed to manage themselves.

So the first step of being a successful self-taught programmer is to master the art of personal management. That means focusing:

- Planning
- Prioritizing
- Executing
- Accountability

**Planning** is critical to effective self-management because without it you are simply hoping that everything just falls into place. As they say; hope is not a good strategy. So make sure to spend a little time every day planning out a.) when you're going to study and b.) what you are going to do.

How are you going to decide what you are going to do? That is where you have to learn to **prioritize**. Prioritizing, in its essence, means that you are declaring that which is most important. It is tempting to choose that which is most fun or stimulating when we study…hence why it is so easy to watch a tutorial. But it may be more important to work on a side project. If you have set your priorities for the day you will know what you need to do instead of what you feel like doing.

Planning and prioritizing are nothing without the ability to **execute**. If you've set up a plan for the day and prioritized that which is most important, you still must be able to execute and actually do what you've set out to do. There's no magic trick for this; taking action on what needs to be done requires that you push yourself when you don't feel like it. You may not be very good at executing tasks now but so what? You'll need to start from there and work on it little by little. Execution is where the rubber meets the road between setting yourself up with a plan and reaching your goal of becoming a programmer.

Lastly, an effective personal manager has built in **accountability** into what they do. This means that they are relentlessly assessing how effectively they are managing themselves and making sure that when they begin to veer off the path that they recognize it. This could mean having an accountability buddy; someone who you meet regularly with and share your successes and failures. It could also mean journaling every day and recounting what went well and what could have been improved. This regular review process solidifies good habits and helps to remove those which are not helping you.

To sum up; the fundamental skill to learn to code effectively is to develop strong skills in managing yourself.

The more you work on planning, prioritizing, executing and accountability the more benefits you will reap.

# LEARN "ACTIVELY"

*"Programming is not a spectator sport".*

*In other words; you can't become a programmer by watching someone else write code. At the end of the day if you want to be a professional software developer you've got to write code....and lots of it!*

This act of writing code, building a portfolio project from scratch or working on a programming challenge is what I call "active learning". This type of learning is where you are engaging you creative muscles and current level of knowledge to solve problems. It is often uncomfortable because no one is telling you what to do but this is where most big breakthroughs in understanding and skill come from. For this reason most people often will avoid engaging in active learning as much as possible.

The other side of learning is what I call "passive learning" where you are sitting back and passively consuming a tutorial or book. This type of learning is easier in the sense that you can sit back and consume information fed to you by an author or instructor. This type of learning feels good because it's what we're used to from school and it feels like it's progress. And while passive learning has its place, many people get stuck in a constant cycle of watching tutorials called tutorial hell. Because a person has become comfortable with consuming information passively, they begin to avoid the one thing that could help them progress: active learning.

But you may wonder; **how much time exactly should I spend on active vs. passive learning?**

With many of my clients I have used a rule of thumb of 2 active hours for every 1 passive hours.

Keep in mind that this is a "rule of thumb" which means you can always use your judgment here.

So as you proceed from here forward you will want to make sure you are engaging in a lot of "active" learning. The best way to engage in this type of learning is to aim to build a number of projects for your portfolio that demonstrate your coding skills.

For example, there may be times when you are focused more on the fundamentals of a programming language, framework, library, etc. In those cases it can be a good idea to engage in more passive learning.

By following this rule of thumb I have found that many people are able to avoid tutorial hell without a complicated system to implement.

To sum everything up; **passive learning is essential but to be a programmer you have to create projects and write a lot of code.** Make sure you're spending a significant amount of your time performing active learning.

# REVIEW

*I believe that the hardest part about being a self-taught programmer comes from the lack of visible progress as time goes by. There will be long stretches of weeks or months where, despite endless hours of practice, there is no evidence that you have improved in any meaningful way.*

This is why so many people quit trying to be self-taught. Being able to see progress is one of the biggest motivating factors in human psychology; when you see yourself moving closer to a goal you feel more motivated.

The frustrating truth about learning to code is that progress is happening even though most of it is not visible to the naked eye. When you are going through the motions of studying you are building up skills **incrementally**.

This means you may be reinforcing ideas, storing things in long-term memory, learning how to creatively problem solve, building mental models and more. It won't feel like you're getting better because the progress is so gradual; you can't see progress until you step back and take a look at the big picture.

Knowing that progress is motivating and that learning to code doesn't provide much visible progress it's on you to make sure you are setting up a system where you are actively measuring and reviewing your progress.

*But how do you track progress?*

One of the easiest ways to track progress is to use a time tracker. A time tracker allows you to see how much time you've put in yesterday, this week, this month and even more. Nothing is more satisfying then looking back on your time tracker and seeing consistency and effort over a long period of time. Working with clients, I have found this to be one of the most powerful tools to view progress.

Another approach that I have personally found is to journal about the learning process on a regular basis. It may sound counter-intuitive but what I have found is that just jotting down your struggles and celebrations in a journal for five minutes a day can give you a ton of insight later on when you look back at it. When you do have the chance to reflect on prior journal entries you will see that some of the things you didn't understand are now things that seem comically simple. This is the easiest way to see progress.

Lastly, the approach I like to take with clients to track progress is to slowly increase the challenge of what they are learning or doing. That could meaning giving them more challenging concepts to learn or projects that are increasingly complex. As they see themselves move from simple to intermediate to advanced they know they are making progress.

The key takeaway here: progress keeps us going and motivates us. Regularly review your process to keep the motivation and enthusiasm high.

# RECOVER

*If you're serious about achieving the highest level of efficiency with your study time then you need to focus on rest and recovery.*

Basically think of it like this; Olympic athletes train extremely hard but then focus on making sure that they recover so they can train hard again the next day.

You want to think of yourself as an Olympic athlete but instead of training your body, you're training your mind.

Many aspiring self-taught programmers make the mistake of stretching themselves too thin. They stay up late, drink caffeine, cut back on sleep, don't take breaks while they're working and so on...all in an effort to "hustle" and "grind".

One of two things happens with the hustle and grind approach. Either the person studying begins to have a diminishing level of returns on their time (meaning they simple stop learning after a certain point) or they burn out completely and end up taking weeks off....or even worse they quit altogether.

To avoid either of these fates you want to make sure that you are getting proper rest and recovery as you proceed.

Rest and recovery to me are three things; **sleep, taking breaks and downtime**.

For the first point; getting sleep means going to bed and waking up at the same time every day and getting your full amount of sleep in (which means about 8 hours). A lack of sleep effects your memory, mood and even impairs your motor skills. This is the most important and hardest thing for most people to regularly do.

Second; taking regular breaks during the study process will help keep you fresh and engaged. You should have regularly scheduled break periods during your study times. So if you study for 30 minutes you should stop and take a 5 minute break...or if you want to go longer then do an hour of studying with a 15 minute break.

Taking these breaks will help give you retain more information and keep your enthusiasm going. On the flip side, when you study for 2-3 hours straight it will often result in a diminished ability to focus and will drain you faster of your energy.

Lastly, make sure to have planned downtime in your schedule. For you overachievers out there you're going to devote every last minute of your time to coding. You're going to probably obsess over it and think about it constantly.

The key is to make sure that you have time throughout your day dedicated to something that is not programming. So that could mean carving out time for exercising, family, hiking or anything else that truly refreshes you.

To summarize the above points; **treat yourself like a mental athlete and prioritize rest/recovery**.

# ABOUT ME

Hey there!

Thanks for checking out this manual! My genuine hope is that it helps improve your study efforts and gets you to your goals faster.

If you aren't familiar with my work I figured I might as well introduce myself.

My name is ***Andy Sterkowitz*** and I'm a self-taught software developer with over 5 years of experience. I landed my first programming job in 2015 after a year of studying on my own. To say that I came from a non-technical background is an understatement: I had worked as a Barista at Starbucks, a server at multiple restaurants and even a car salesman.

During my career I've worked for two companies as a software developer and even founded my own startup (called CaseCentric) which didn't end up getting off the ground. I've worked with many different programming languages and technologies but my favorite by far are C# and Javascript.

In addition to that I also have a YouTube channel with over 225,000 subscribers (at the time of this writing) where I share videos on how to be a self-taught programmer. I currently run a mentorship program where I work with aspiring self-taught programs to coach them on their way to getting into the industry.

# READY TO BECOME A SOFTWARE DEVELOPER?

If you've committed to the journey of becoming a software developer and want to work with me to reach your goals, go to *https://andysterkowitz.link/coaching-call* and book a FREE career strategy session.

During that call we'll figure out whether the mentorship program is a good fit by looking at what where you want to take your career as well as getting clarity on what's been holding you back. From there; if the mentorship program is a great fit then we can explain the program in detail and talk about how to get started.

## No strings attached, the call is 100% free.