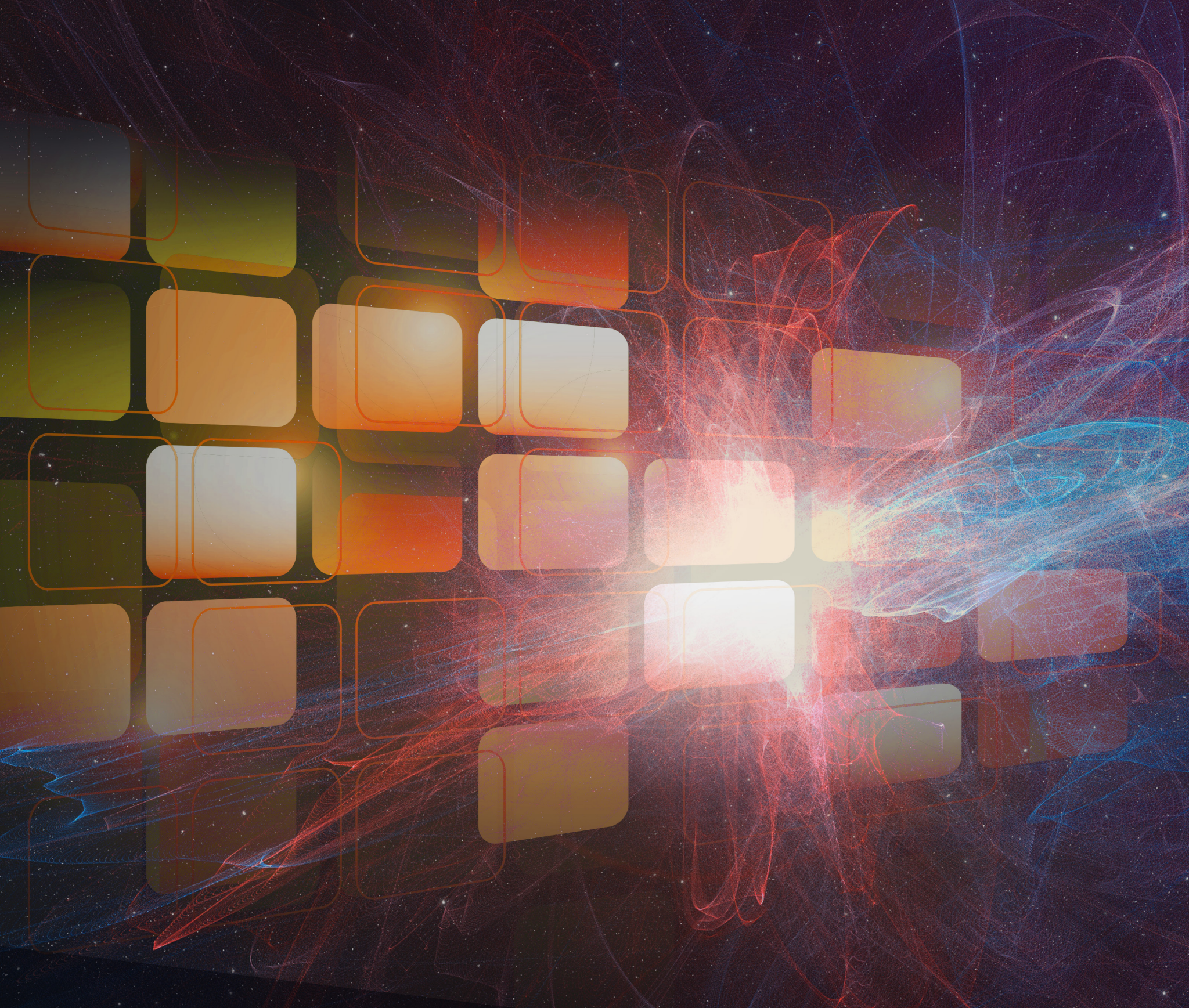

10 Things Software Developers Should Do Every Day

TRAVIS NELSON https://travis.io

— THE PRINCIPLES

10 Things Software Developers Should Do Every Day

TRAVIS NELSON

— ABOUT THE AUTHOR

Travis Nelson is a consultant, software engineer, and architect specializing in server and client-side development, project management, and development practices.

He has a passion for helping developers and teams get laser focused, communicate clearly, and build better software.

He has professional experience in marketing and graphic design, software development instruction, and he's fanatical about personal growth and opportunity creation.

He also likes to speak about himself in the third person.

https://travis.io
me@travis.io

“How you do one thing is how you do everything.”

Being a great developer isn’t about writing code. At some level you already know this. If you’re anything like me, though, you spend your days writing code, but often feel disconnected from the bigger picture of why you’re writing it.

I created this guide for two reasons: First, I wanted to document some things that I’ve seen excellent developers do. These are the highest-level technical architects, engineers, and leads who set the bar in the best projects I’ve worked on—in the enterprise, in successful startups, in innovative tech companies, and in premier consulting firms. And truthfully some of these principles are ones I’ve seen the worst examples of; in those cases the writing here is just as inspired.

Second, some of the practices here are ones I also want to improve. These are principles I value, but some days I’m far from a shining example of any of them. In my experience the best way to learn something is to teach it, so consider these principles something we should all strive for, myself included.

To being your best self, and a better version of it every day.

Travis Nelson
https://travis.io

— ONE

Speak Up

Staying quiet and being a fly on the wall in meetings doesn't help anyone—especially you. Speak your mind, ask stupid questions, and be a part of the conversation in every meeting.

Don't sit quietly in meetings and let the most vocal team members drive the conversation. Speak up and let your ideas be heard. Be confident in your knowledge and know that your contributions (even bad ones) make the team better in one way or another.

Don't be afraid to ask stupid questions. If you don't know something, and it's important that you do, inquire. If it's not appropriate to interrupt a meeting to get clarification on something, use a messaging tool on the side to ask someone who knows, or make a note to get clarification later. You never know when you'll be called on to engage in the conversation, so remain plugged in.

Be an active part of the conversation and don't let your ego get in the way. Not only does it show that you care about contributing, but it shows that you're engaged and not just going through the motions.

When it comes to paying attention in conference calls, a wireless headset is a game changer. Personally, I get up and walk around, because it prevents me from doing work or getting distracted when I should be paying attention.

— TWO

Be a Problem Solver

As you mature, your knowledge of a specific language or technology becomes less important; building software is about solving problems, so be good at that.

No matter how boring or annoying it might be (just being honest here), part of your job as a software developer is to understand not just the technical domain of your project, but the business domain as well. This becomes more important the further you advance in your career.

Your real job is to *solve problems*, and those problems have a direct impact on the business and ultimately the bottom line of the company (or client). You're a much better developer when you understand the actual problem you're solving. And truthfully, having a more holistic understanding can make the work more interesting, and give you the satisfaction of being a vital part of the project than just a cog in the machine.

Get into the habit of understanding the impact your code changes have on the business, as disconnected as it may seem at times. Once you learn to focus on the value you're working to deliver, the technical details usually fall into place much easier.

— THREE

Review Your Approach With Others

Features or bugfixes in code can be implemented several different ways. Some ways are better than others; be open to hearing recommendations to find the best approach.

As you progress through your career and work on different projects, you'll find there are many ways to solve the same problem with code. While the end result might be the same, there are a few things to consider in your approach when writing a feature or bugfix. Code reviews should be part of your process, but prior to that, a simple discussion to vet your approach in advance is always beneficial.

Is your approach performant? Other developers might have good suggestions to avoid performance bottlenecks, or tweaks that allow your software to work asynchronously, or better handle horizontal scaling at the infrastructure level.

Is your approach secure? Developers more familiar with security, cryptography, or even your company's infosec team policies might help you avoid dangerous gaps that would allow malicious users to compromise your system.

Is your approach a good fit? If your approach doesn't match the overall theme used throughout your project, it may be hard for other developers to use or update it when needed. Whether you're writing pattern-heavy enterprise software, or pragmatic, beta-level startup apps, your changes should blend in.

— FOUR

Take Excellent Notes

Most developers take too many mental notes and not enough written ones. Build a habit of writing things down for your reference, and for your sanity.

Note-taking serves two main purposes. First, keeping good, detailed notes helps you recall things more accurately (obviously). But a much-underestimated benefit of habitually taking notes is the relief it gives your already-full brain.

Developers have a bad habit of walking into meetings without some way to take notes and making a lot of promises to “look into that” or “follow up with so-and-so”. But these promises often fade into the ether if they’re not tackled immediately after the meeting (or during it); sometimes until we’re reminded of it days later by a manager who needs a status update.

Keeping everything in your software developer brain is impossible, so don’t try to force it. Redirect to-dos and other notes to a notebook (or a desktop app like Microsoft OneNote - https://onenote.com) and don’t allow them to take up valuable space in your head—and eventually get you into trouble.

Most importantly, have a good system for reviewing your notes, making sure nothing gets lost in the shuffle. I use and highly recommend Bullet Journaling (https://bulletjournal.com/pages/learn), but there are other systems that might work for you.

— FIVE

Stay Plugged In and Visible

Gone are the days of large, poorly-managed software teams. Modern teams (especially remote ones) require more communication and connection; not less.

Anyone who's been on a poorly-managed software team knows how easy it can be to fade into the background and browse to the end of the Internet on company time without anyone being the wiser.

Modern teams, however, are more in tune and fluctuations in productivity are much more noticeable. Practices like Agile make developer contribution (and lack of it) fully transparent, and it's much harder to be one of those head-down developers who's always "really busy" but not producing anything.

Energy comes and goes, and good teams know this. But find your cadence and try to stick to it. Communicate openly with your team, have conversations in public channels instead of private ones, vet your ideas with other developers and architects, and be involved in code reviews and demos.

In the short term, by being plugged in, you'll learn more and you'll build trust and rapport with your colleagues. In the long term you'll benefit your career in unexpected ways, and you'll build communication skills that serve you both personally and professionally.

— SIX

Take Ownership

Don’t sit back and let someone else take the glory or the blame. Be the brave one; take ownership of things good and bad and you’ll earn both accolades and opportunities.

It takes a strong person to accept blame for something they were responsible for, but it takes an even stronger person to accept blame for something that they weren’t responsible for, but could have been.

The most mature software teams are those where the source of a problem usually unfolds with someone saying something like “I’m sorry, that was completely my fault.” It seems easy enough, but consider how often you hear blame directed at others. “The environment was down”, “so-and-so didn’t do his/her job”, “I didn’t have the resources I needed.” Blame is a by-product of fear; fear of reprimand, fear of looking stupid, fear of losing respect.

Be known as the person who won’t let things slip. Reach out to your coworkers and make sure their needs are covered. Take an opportunity each sprint retro (or on a schedule if you’re not part of an Agile team) to review one area where you feel you could improve, and share it with the team if it could benefit them too.

Lastly, if you make a mistake it’s better coming from you directly, and at the earliest opportunity. Being upfront about it reinforces trust and prevents the embarrassment and shame of someone else discovering your mishap.

— SEVEN

Socialize Knowledge Early and Often

Gatekeepers, those who solely hold specialized knowledge, are painful to have and more painful to lose. Prevent gatekeepers by fostering routine knowledge sharing.

There's little worse than losing an employee who holds specialized knowledge about your projects, or who's maintained important client relationships in your company. The impact is highly stressful, with remaining staff working long hours rebuilding client trust, all the while asking frustratingly basic questions they shouldn't have to, or even introducing bugs due to a lack of understanding.

Knowledge should be socialized constantly. Maintain a centralized tool or repository for documentation in an easy-to-edit format like Wiki or Markdown, and keep it up to date. New team members should be trained to reference it, and veteran team members should review it often for accuracy.

When writing code in unfamiliar areas, reach out to others as soon as the need arises, and do so in public forums rather than private, when appropriate. Making these discussions visible helps ensure you don't become a gatekeeper yourself, and it helps others become more cross-functional.

Make a habit of commenting your code in a way that would help someone understand its intention (not necessarily its implementation), even if they've never seen it before.

— EIGHT

Help Others

There's a stigma around asking for help, and many developers hesitate to. Be a champion of supporting others; it helps your team succeed, and the personal benefits can't be ignored.

There are myriad ways to help your colleagues and intentional, helpful outreach goes a long way—for your team, and for you personally.

Make routine contact and find ways to help your coworkers. Create automation scripts that save time and share them with team mates who can benefit from them. Offer your time to help solve problems, or when a fresh set of eyes might be useful. Remember what they're working on and check in to make sure they're making progress. Offer to take on tasks like writing unit tests or validating their changes, when you have the bandwidth to do so.

It's worth noting: whether you want to believe it or not, people are going to talk about you when you're not in the room. Without a good, positive precedent, many conversations about other people end up neutral at best, and negative at worst. We're humans; we talk shit about one another sometimes. Investing in others helps maintain rapport during stressful times or when you make mistakes.

Take control of your personal narrative and position yourself as dependable and helpful, and an asset to your team.

— NINE

Communicate Effectively

Email and chat are great for archive-ready discussion, but not always clear. Just as a picture is worth a thousand words, a phone call or screen share is worth a thousand messages.

For much of my career I considered myself an effective online communicator, able to understand and explain complex ideas via chat or email. Later in my career I was put on a team that communicated mostly via Skype calls and screen shares, and it changed the way I think about communication within teams.

In retrospect I learned that text-based communication is slow, one-sided, and highly nuanced—largely because it removes the human element. A person's phrasing in a chat message might make sense to one person, confuse a second, and outright trigger a third. Voice communication adds inflection, timing, emotion, and conciseness that just can't be matched in chat or email. Video takes this a step further (though in my experience, the benefits aren't as drastic.)

Many developers seem to be uncomfortable with the idea of "talking on the phone", partly because it's a context switcher, and because it forces on-your-feet thinking. Embrace this discomfort and push yourself to do it; as [anti] social networks and YouTube comments drive us further away from real conversations, be the one who prioritizes phone, video, and in-person discussion to *connect* with others. The benefits are immeasurable.

— TEN

Network and Connect With Others

The most well-rounded and most sought-after developers are those who invest in themselves—and not just to improve their technical knowledge and skills.

For better or worse, much of your professional growth has to do with how well and how often you connect with others. Your skills are important, but your connection with others will ultimately determine your career path. Seek out people in your organization and industry who you can help, and who can help you keep moving in the direction of your goals.

Within your workplace, reach out and introduce yourself to architects, managers, and other project leads, and present yourself as someone who is willing to be a bridge for communication across (and within) teams. Outside of your workplace, connect with everyone you can on networks like LinkedIn, Alignable, GitHub, and even more personal social networks like Facebook and Twitter.

Always dress professionally, and here's a tip: keep your profile photo up to date (Skype, Slack, email, etc.) with one that actually looks like you—professional and recognizable. Ideally, people you communicate with online before meeting you in person should recognize you immediately.

Lastly, any time someone helps you, be sure to reach out and thank them.

— THANK YOU

Thank you for taking the time to read this guide. To learn more about me and what I do, see the links below.

For information about 1:1 Developer Coaching, visit:
https://travis.io/coaching/

Want to hire me to help make your project a success? Check out:
https://travis.io/hire-me/

Connect with me on LinkedIn:
https://www.linkedin.com/in/travisneilnelson/