# The Definition of Senior: A Look at the expectations for Software Engineers

March 26, 2023 **#life**, **#career** No comments yet

— Published by  Luciano Mammino

In today's fast-paced tech industry, the role of a senior software engineer has become increasingly important and something many engineers strive to be recognized as. But are you senior yet? And what's expected from you a senior?

The truth is that answering these questions is a very subjective matter and different people or different companies might give very different answers.

In this article, I'll try to give you my take, based on my opinion and personal experience.

So let's get to it!

> **This article is based on a presentation I gave last year called The senior dev, an opinionated take.**

Hopefully, this article can inspire engineers to grow in their careers, help managers to hire the right people, set expectations, and support their teams.

## Senior? Yes, but how much? 💪

The first remark I have to make is that I am not going to focus on different levels of seniority like what it means to be a *Staff*, a *Principal* engineer or even an *Architect*.

What I want to focus on is what it takes to be considered and valued as a *Senior* in the most general sense.

Ok Luciano, but what heck do you mean by *being senior in the most general sense*?

The core responsibilities of a senior engineer are to **move projects and people forward**.

Some people like to call that "*being **a force multiplier***", which, be aware, is very different from being a *10x engineer*. In fact, a senior engineer is first of all a **team player** and not a lone hero, or a rockstar, a superstar, a magician, a grey-bearded wizard, a unicorn, or a *whatchamaccalit*!

Easier said than done! What kind of skills, mindset, and duties are we talking about? How can an engineer effectively help to move things forward (more than a junior or a mid-level engineer can do)?

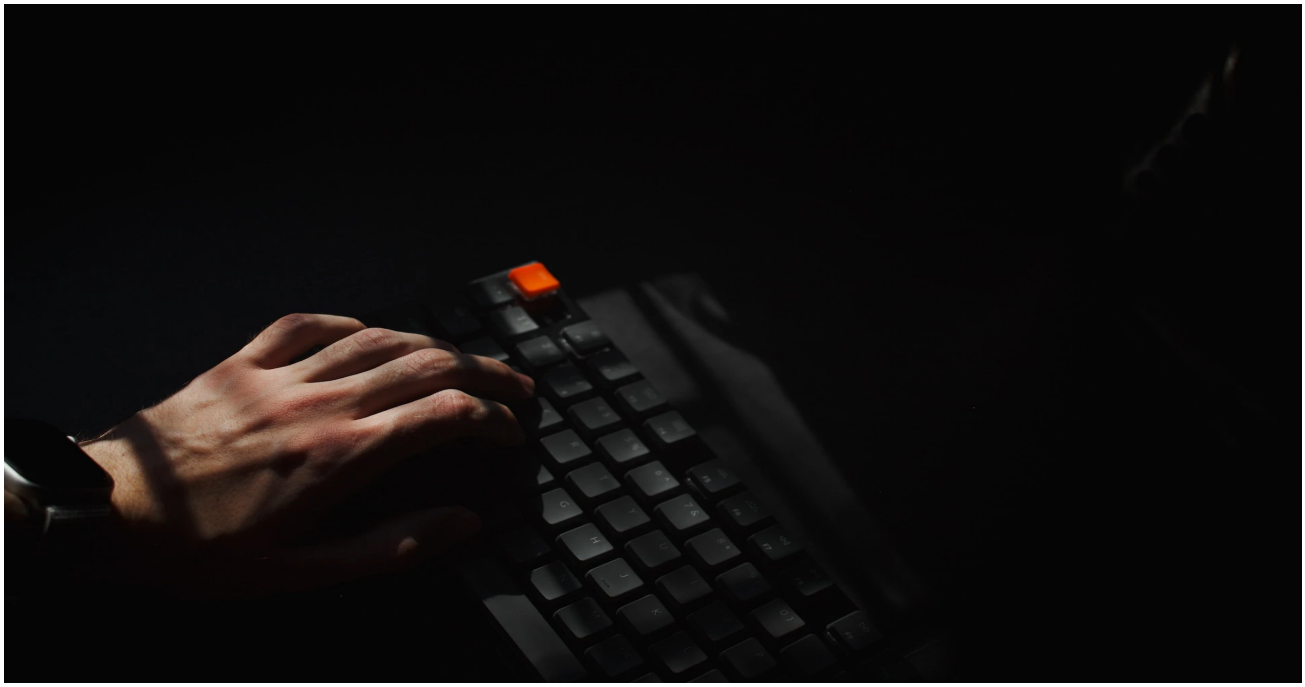

Photo by Quinton Coetzee on Unsplash

## Am I a senior yet?

The first myth to bust is that being senior is not necessarily a matter of how long one has been in the market!

> **More time ≠ more senior**

Yes, it definitely helps to have been around for longer. The more you do something the higher the chances that you will have faced different challenges and learned something from them!

But let's not say that you are a senior if you have *5+ years of experience* or something like that. Unfortunately, this simple model won't work well for many engineers!

Similarly, being a senior is not a matter of age. Growing older doesn't necessarily make you better at your job... a senior is not someone with *at least 40 years of age*...

In my career, I had the fortune to truly appreciate this.

I had some very young but extremely passionate and skilled colleagues that I would definitely consider senior, and I also had *older* colleagues with many years in the industry on their CV but their contribution to the team and the project was nearly close to zero, if not even negative! I wouldn't have called them senior, even though, most of the time, the company recognised them as such...

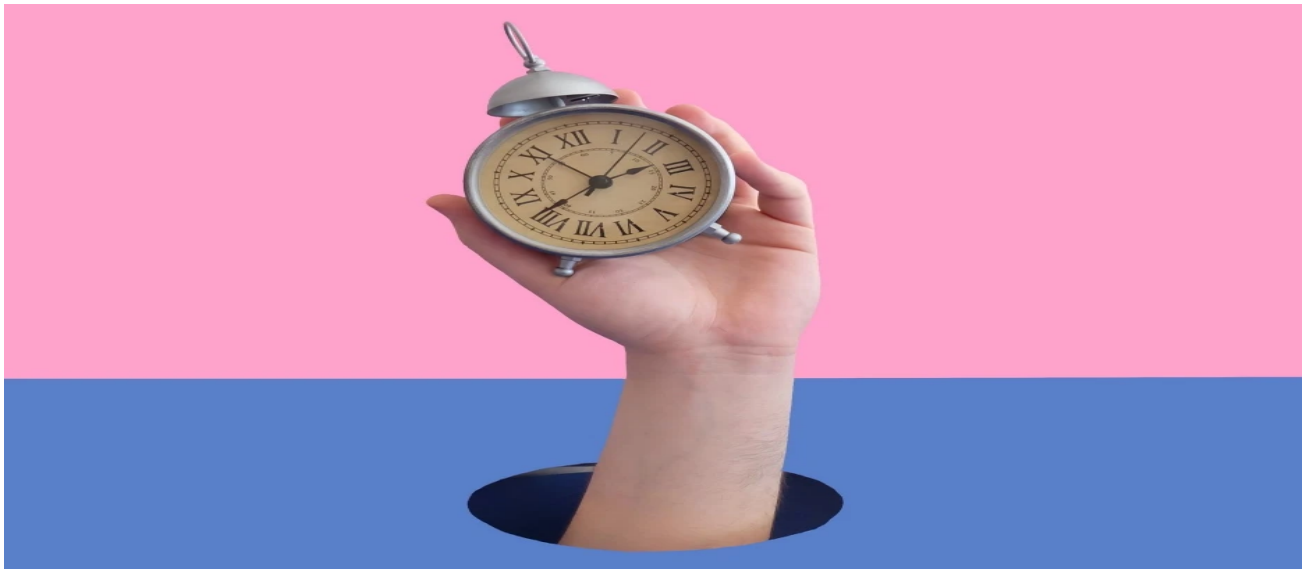

Photo by Elena Koycheva on Unsplash

## What skills do I need?

If it's not a matter of time in the industry, what is it then?

Remember, it's all about moving your project and your team forward... and this takes some skills, so what kind of skills are we talking about?!

When it comes to **technical skills**, a senior software engineer is expected to have a solid foundation in programming languages, frameworks, databases, architectures, etc. However, technical skills are not the only thing that sets a senior software engineer apart. **Soft skills**, such as communication, leadership, and problem-solving, are equally important.

I would even argue that **for a senior engineer, soft skills are even more important than technical skills**, because it's not just about what you know, but is about how you communicate that, how you influence other people to leverage your knowledge, how you share your expertise, unblock tricky situations, and make the team more self-sufficient and productive.

Communication skills are vital for senior software engineers, as they often work with cross-functional teams and communicate technical concepts to non-technical stakeholders. A senior engineer must be able to clearly articulate their ideas, listen actively, and provide constructive feedback.

Leadership skills are also essential for senior software engineers, as they are often responsible for mentoring junior team members, leading project teams, and making hard technical decisions. A senior engineer must be able to inspire and motivate their team, delegate tasks effectively, and provide guidance and support when needed.

Problem-solving skills are another critical skill set that senior software engineers must possess. They must be able to think critically, analyze complex problems, and find creative solutions. Senior engineers should be comfortable with ambiguity and be able to adapt quickly to changing requirements.

At the end of the day, software engineering is all about solving problems!

But in complex organisations, with many stakeholders coming from very different backgrounds, problems are not only technical. And problems can be badly defined and change often. So, being able to contribute to understanding what is the problem at hand, creating and sharing a vision, steering the team to take the right direction are all crucial skills that can have a massive impact.

The best *technical* solution is not always the absolute best solution for an organisation.

The best solution is one that accounts for how easy it is for more junior people to contribute to it, how easy it will be to evolve it and adapt it to changing requirements, how well will it scale if the business is successful, etc.

Sometimes you might have to compromise on purely technical qualities to come up with solutions that are optimal for the team and the business environment you are working on.

It takes a lot more than just technical skills to see all the facets of a project and be able to come up with solutions that can fulfill all the needs.

Photo by Kelly Sikkema on Unsplash

---

## Am I technical enough?

Ok, ok... but I know we should really talk more about pure technical expertise!

After all, given the way the tech industry works today, this is the first thing people will recognize you for!

If you cannot demonstrate technical expertise you are not even going to have a chance at getting a spot for that job position you really like!

## T-shaped profile

As a senior software engineer, possessing technical skills that span across different domains is crucial to thriving in the ever-evolving landscape of software development. To be an effective and efficient senior software engineer, a **T-shaped profile** that emphasizes mastery of one skill and proficiency in others is essential.

Photo by Jonathan Farber on Unsplash

A T-shaped profile for a software engineer refers to having expertise in a specific skill or domain (represented by the vertical stroke of the T), while also having a broad understanding of other areas (represented by the horizontal stroke). This allows individuals to contribute to a project in multiple ways while still being an expert in their primary area of focus. T-shaped profiles enable effective collaboration with team

members who have different skill sets, leading to a more successful development process.

A good way to build a T-shaped profile is to start by developing your vertical stroke: what will eventually become your core strength!

Focus for a while on one area only. Yes, it's better if you pick one that you really like.

When you feel you mastered that particular domain try to expand your knowledge around it and start developing your horizontal stroke. Step outside your comfort zone and explore related topics. Ideally, these related topics should compound your existing knowledge and extend your ability to build products end-to-end.

If you have become an expert in API development, you might want to explore databases or frontend development, because those will start to move you to a more "full-stack" profile. Similarly, you could also explore infrastructure-as-code and learn how to bring applications to production.

The trick is not to try to become an expert in every one of these additional areas, but just to learn enough to be able to understand the basic ideas and collaborate effectively with people who have these areas of knowledge as their core expertise.

### Broad understanding

A broad understanding of software development is also necessary. This includes understanding the platform, architecture, code structure, testing, deployment processes, and scalability models.

Moreover, a senior software engineer should comprehend the tradeoffs between different technologies and paradigms such as Monolith vs Microservices, Memory vs CPU, Highly Scalable vs Low Latency, Reusable vs Bespoke, and Complex vs Simple. Understanding the short and long-term impact of these tradeoffs is also crucial.

Developing this kind of knowledge is non-trivial. It certainly requires spending a significant amount of time building products and doing so adopting different practices, programming languages, frameworks, methodologies, etc.

You can broaden your knowledge only if you appreciate that every technical choice comes with its own set of tradeoffs. There will always be positive aspects and negative ones.

### There's no silver bullet in technology!

There's no framework that you can use to build any kind of application optimally!

If you have a comprehensive enough understanding of the tech landscape, you should be able to tell which set of technologies is the best to address the specific problem at hand.

And, of course, it's ok to be proven wrong, as long as you keep an open mind and you leave yourself (and the team) room for making mistakes, learning from failures, and trying alternative approaches.

The path to success is rarely a straight line...

Photo by Sylvain Gllm on Unsplash

## Gonna catch all bugs!

We know that bug-free software is a myth, so I am not going to say that a senior software engineer doesn't create bugs! Of course, they do...

But it's really important for a senior software engineer to know what can be done to reduce the number of bugs to the very minimum and to try to spot them as early as possible (possibly before the users do).

Understanding and refining user stories, writing different types of tests such as unit, integration, and end-to-end tests, finding and discussing edge cases, and keeping track of technical debt are all essential skills.

A senior software engineer should be able to make a big difference here and help the entire team to write better software by introducing and educating best practices about software testing.

I would argue that it's not too hard to write tests, but writing good tests it's almost an art!

To write effective tests, you need to write testable code. This means designing code that's modular, loosely coupled, and has well-defined inputs and outputs. The better the code is designed, the easier it is to write tests for it.

Your tests should be easy to read, understand, and maintain. Use descriptive names for your tests, and write tests that cover one specific feature or behavior. Avoid testing multiple features in the same test, as it can make it harder to pinpoint the source of the error.

But how to learn to do all of this?!

Again, practice practice practice!

Start by accepting that untested code is bad code (or to the very least code that should keep you awake at night), then try to understand the specific business domain and what makes sense to be tested and how: what kind of inputs are the most representative, what kind of outputs are expected and what kind of side effects can happen.

Learn the different types of tests and different testing frameworks. Finally, learn which types of test will provide the most value for the business and optimise the team workflow to focus more on those.

Photo by Charlotte Descamps on Unsplash

### Promote the right patterns

Having written a book on design patterns I am certainly biased on this one...

A senior software engineer should have developed some degree of *flexibility* in different programming languages and they should understand paradigms such as OOP vs Functional, Declarative vs Imperative, and Compiled vs Interpreted. This will allow a senior software engineer to be able to solve the same problem using different tools and techniques.

Similarly, a senior software engineer should understand various design patterns and best practices. This will allow them to suggest patterns that can have good long-term effects, and avoid patterns that will eventually lead to problems.

Code is something that will need to evolve as the business and customer needs change, so it is important to design with evolution in mind.

Patterns are not silver bullets, but just blueprints that you can decide to use to address common situations and pave the way for future improvements.

Don't try to be dogmatic and apply specific design patterns *just because you can*, try to think about what's the value that the specific pattern is bringing to the project and whether that value is actually important.

Sometimes, *simple dummy code* can be very effective and it can be understood and evolved easily. Some other times you do need the structure imposed by specific design patterns because that might able to make your code more testable, extendable, configurable, etc.

As a senior software engineer, this is an opportunity to shine and have a significant impact during pairing sessions and code reviews!

---

## What kind of soft skills?

Now that we covered which technical skills will give you an edge in your career and allow you to be recognised as a senior engineer, let's address the elephant in the room: soft skills.

You got that interview thanks to your hardcore technical skills, but now you have a chance to really make an impression by showcasing a range of soft skills.

## Growth mindset

The mindset of a senior software engineer is just as crucial as their technical and soft skills.

A senior engineer must have a **growth mindset**, a willingness to learn and adapt, and a drive for continuous improvement.

They should be **comfortable with failure**, see it as an opportunity to learn, and **be willing to take risks**.

Always remember a senior engineer must also be a team player, not a hero. They should collaborate with their team, support their colleagues, and contribute to the team's success.

## Being an active lever

A senior software engineer should have a very proactive role in an organisation. They shouldn't just isolate themselves in a room and keep smashing their fingers against the keyboard.

A senior should know when it's time to ask hard questions and take leadership to find what they don't know.

Being able to bring a strong technical perspective to a business conversation can be very impactful and inform the business on what's the best strategy to move things forward.

To all effect, a senior should become a bridge between product and technology.

This sometimes means that a senior should also **know when to say NO**. No to quick and dirty solutions (that will eventually backfire hard) just to hit a deadline. No to plans that

only account for building features on top of features without considering user experience and the long term stability of the product. No to arbitrary technical choices just because "*we have always done it this way and it has been fine*".

Saying NO is easy, but also very hard. Everyone can just say NO, that's the easy part! What's difficult is to argue the why of that NO, propose alternative points of view, find compromises and defuse short or long term time-bombs.

Being effective at this requires a fine degree of soft skills. You should know how to communicate effectively and pick your battle wisely. You cannot always say NO to everything or you'll become recognised as a grumpy *naysayer*.

It's more about being able to say *"NO, BUT"* rather than just a dry *"NO"*. And you should also be capable of doing that in front of the right people and at the right time...

When done successfully, this can have a massive positive impact on the business and the team. It can lead to innovation and create new business capabilities or unique competitive advantages.

This is an area of growth for me. One of those things where, when I look back at my career, I feel like I have failed at it many many times. But hopefully, I have learned something and over time I'll get better at it...


Photo by Michał Parzuchowski on Unsplash

## Understanding the business

To be an active lever in your organisation, you need to be able to understand the business as deeply as possible.

Do you know what's the purpose of the business? What's the long-term vision? Is there a clear strategy? What are the unique strengths and what are the weaknesses?

If you can answer all these questions you are in a good position to use your technical skills to determine how technology can help the business to succeed.

Only with a solid understanding of the business you will be able to pick the right battles, focus on what matters and help design systems that can serve the business well today but also be adapted to future needs.

## Communication

> **"Good communication is as stimulating as black coffee and just as hard to sleep after." – Anne Morrow Lindbergh**

As a software engineer, communication skills are crucial to your success. You must be able to communicate effectively with a variety of stakeholders, including other engineers, project managers, customers, and executives. This requires a wide range of communication skills, from speaking and listening to writing and presenting.

Since you will need to talk with all the stakeholders, you should learn how to communicate technical concepts to non-technical stakeholders in a way that is easy to understand. It also means being able to listen to feedback and incorporate it into your work. The ability to communicate effectively with stakeholders is essential for ensuring that everyone is on the same page and that projects are completed successfully.

Another important communication skill for a software engineer is the ability to explain uncertainties and propose ideas on how to address them. Software development is an inherently uncertain process, and there will always be unknowns and unforeseen challenges. As a software engineer, you need to be able to communicate these

uncertainties to stakeholders in a way that is clear and concise. You also need to be able to propose ideas on how to address these uncertainties and move the project forward.

In addition, software engineers must be able to talk about failures and learnings. We already covered this, but it's important to remark that failure is an inevitable part of software development, and it is important to be transparent about failures and the lessons learned from them. This requires effective communication skills, including the ability to take ownership of mistakes and communicate them honestly to stakeholders. It also requires the ability to communicate the lessons learned from failures and incorporate them into future projects.

Communication skills are also essential for writing documentation and delivering presentations. Software engineers must be able to write clear documentation that explains how their solutions work and how they should be used and operated in production. They must also be able to deliver presentations that effectively communicate technical concepts to a non-technical audience. This requires strong writing and presentation skills, as well as the ability to simplify complex topics and make them digestible for the audience.

Good communication skills can open many doors and unblock complex situations. So make sure to practice them as much as possible!

### Supporting management

When you are a good communicator, you become someone who can be very effective at supporting various management activities, for instance, planning and driving ceremonies. You can help with keeping track of priorities and technical debt and split complex tasks into manageable parts.

You are in a unique position to understand and leverage team strengths and pull in the right people at the right time.

By working closely with management, you can ensure that your work aligns with the overall business objectives and that your team's efforts are channeled in the right direction.

Planning is another crucial skill for a software engineer. This involves breaking down large projects into smaller, manageable tasks, estimating the time required for each task, and

prioritizing them. Effective planning helps to ensure that projects are completed on time and within budget.

Driving ceremonies such as stand-up meetings, sprint reviews, and retrospectives can provide an opportunity to share progress, identify issues, and plan for the future. As a software engineer, you need to help drive these ceremonies and ensure that they are productive and efficient.

I am not suggesting that senior software engineers should replace the role of project or product managers, but instead, they should complement these roles and provide their unique technical perspective to keep objectives, plans and execution aligned with the expectations of the technical team.

Photo by Annie Spratt on Unsplash

## Autonomy and focus on delivery

The last 2 soft skills I want to touch on are autonomy and a focus on delivery.

Sometimes senior software engineers are in a unique position to contribute significantly to projects that require in-depth research and grind. Heavily technical projects such as the ones related to performance optimization, significant refactoring, and re-architecture.

These are often projects that are quite important for the business because they can unblock innovation and bootstrap new business capabilities.

In my experience, it's not uncommon that in such projects, a single individual contributor might be well positioned to spend some amount of time focusing and achieving a significant amount of progress in a very short amount of time driving the first big chunk of the research forward.

But even in these situations, it is important to avoid silos at all costs.

If you are working on such a project, make sure to have regular check-ins with management and other senior colleagues to keep your progress in check, get useful feedback and share your learnings.

But the ability to deliver value by working autonomously is not only useful in research projects, but it's also a good skill to master in general.

In the software industry, it is more common to work with things we don't know than it is to work we things we know!

There are always new domains to explore, new challenges to face and new tools to deal with. Therefore, it's important to be able to learn quickly and upskill fast when needed. You shouldn't be waiting for other people to teach you and tell you how to do certain things, you should be able to do a certain amount of progress on your own, for instance by consuming and understanding documentation and existing code.

If you have broad skills already, it should almost be second nature to be able to add more to them when needed.

Of course, it is OK to ask for help if you feel stuck, but it's also important to try to do everything you can to progress on your own so you can build as much context as possible and maximise the outcome of the time spent receiving help from others.

Ideally, a senior software engineer doesn't need too much guidance. They can figure out what's needed to move things forward by themselves and involve other people when needed.

Another facet of this involves negotiating expectations and understanding what it means to be successful in the current environment. It's good to be upfront with management and the rest of the technical team and define what individual and team success looks like. If you don't know what's expected from you and the rest of the team, even if you feel you

are doing the right thing, you might end up not delivering the value that is expected from you; similarly, you might not be able to help your team at being successful.

Building this attitude of bluntness and clarity generally keeps the mood more positive and helps with avoiding disappointments.

Of course, you can't always avoid all disappointments or failure scenarios. So when the poop hits the fan, you might be tempted to blame someone else or the system, but it's instead more mature to ask what could have been done better and what could be done in the future to avoid similar issues from raising again.

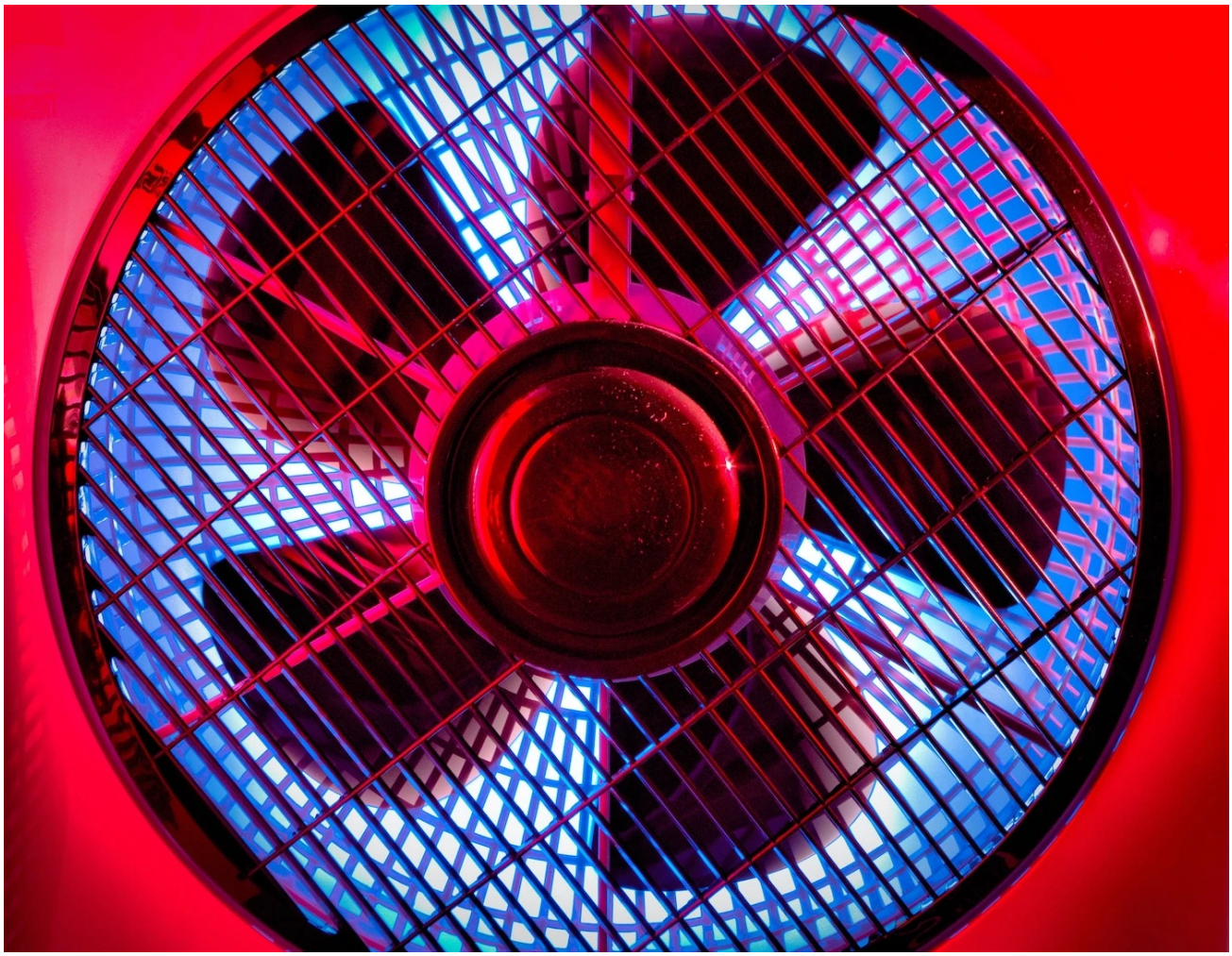

Photo by Immo Wegmann on Unsplash

---

## How can I grow?

If you're a software engineer looking to level up your skills and become a senior member of your team, you might be wondering what steps you can take to get there.

Well, fear not! Here are 4 ideas (plus some extras) that I believe will help you on your journey.

### Go one level deeper

First off, don't just scratch the surface of the technologies you're working with. Dive deeper and learn about the underlying layers.

You have probably built a website or an API, but did you ever wonder how does the HTTP protocol work? Or even how does the TCP protocol works and what happens to establish a connection? Appreciating these details will give you a much richer understanding of the technologies you work with every day.

Yes, it's an endless rabbit hole if you start to dig... so how we dive deeper without getting lost and get value from it?

My rule of thumb (which I have ~~stolen~~ learned from my friend Roberto), is to dig only 1 level down starting from the technologies you are already familiar with.

Have you spent some time with *OAuth* and *OpenID Connect*? Well, you'll probably want to know what's inside a JWT. Do you understand the OAuth authorization code flow and what happens with all the redirects? Actually, do you know how signing algorithms such as RSA and HMAC work?

These are all topics that you can explore to consolidate your knowledge of this particular domain. They will allow you to explore more general concepts that you might be able to re-use somewhere else.

All good steps towards broadening your knowledge and becoming a more well-rounded engineer.

I am often biased toward building stuff. I feel like I don't truly understand something if I can't build a small prototype for it. If you are like me you could try to do that as well. Can you decode a JWT token without using a library? Can you take it a step further and even implement the signature verification algorithm?

Building prototypes is a great way to memorise certain concepts and truly put your understanding to the test. Of course, it takes more effort to build stuff, so choose your prototype exercises wisely: you can't possibly re-implement 60+ years of software engineering just for the sake of learning!

One funny video that I really enjoyed lately is The Computer Science Iceberg. Why do I like it so much? Because it truly illustrates this idea of descending levels of abstractions and exploring more fundamental and generic pieces of knowledge.

Photo by Sergey Pesterev on Unsplash

## Have fun

This is one of my core beliefs: software engineering is a very challenging profession. Not because it's harder than other professions but because things move so quickly that you can never stop learning and you'll always have to feel behind.

The one thing that could make the profession a little bit easier is a genuine passion for the subject. If you are passionate it's going to be easier to motivate yourself to learn new things over and over!

But what if I don't know if I am passionate enough?

Fair, especially if you are at the beginning of your career. There is so much in front of you that it might be scary and even discouraging.

Again, my recommendation is to be biased toward building stuff. Building something will give you tangible feedback that what you are learning can actually be useful. You can build side projects to put new knowledge into practice.

Also, you shouldn't be shy to show what you built (and what you learned) to your peers and even your friends outside of work! Chances are you'll get feedback and come up with new ideas and new things you'd like to learn and try.

Even better if you realise you can apply some of these learnings at work. Maybe you can develop a new path in the company you are working for, maybe you can help with something that is currently being neglected because no one else has the time or the expertise, maybe all of this can demonstrate more value and you'll get a promotion!

When you apply this mindset of continuous learning and sharing knowledge at work, this might generate cross-pollination and you might end up with a team that has fun learning and building together.

Another idea to make people gel with each other and to generate cross-pollination of ideas is to organise company **hackatons** or **free-study days**. These are great ways to help teams to become more passionate about what they do, explore new ideas, learn new things, and ultimately put people in a position to deliver more value in the short and the long term.

### Pair programming

Pair programming is another great tool that you can leverage to give a boost to your expertise.

pairing with as many people as possible within your organization is key. Even if someone is more junior than you, they can still have insights and perspectives that you may not have considered before. Pair programming gives you a platform to share ideas and learn from one another.

Similarly, don't discount the value of teaching others. Even the most senior members of your team can benefit from your knowledge and expertise. By pairing with others, you can share your skills and help everyone grow and develop.

I picked up so many tricks by pairing with other people. Even small things such as how they configured their editor or their terminal.

Now, I get it - not everyone loves pair programming. If that's the case, don't worry! There are other approaches to follow that can still provide you with valuable feedback and help you learn from your peers. Interactive code reviews and show-and-tell sessions are great alternatives that allow you to share your work and receive feedback without having to work together in real time.

In the end, the most important thing is to remain open to new ideas and approaches. By continuing to learn and grow, you'll be well on your way to becoming a senior software developer. So why not give pair programming a try and see what you can learn?

Photo by Janko Ferlič on Unsplash

## Content creation

First off, creating content can take many forms: articles, talks, videos, Twitter threads, you name it! And guess what? You don't need to be an expert to share something new you learned. Even if you're just starting on a given subject, your fresh perspective can bring value to others and help you establish yourself as a thought leader. So do create content on what it felt like to try a new programming language or a framework. What did you like? Was there something confusing? What did you miss from your previous experience?

But here's the thing: you gotta make it a habit. Commit to a regular schedule, and you'll develop discipline and consistency in your content creation. Plus, the more you create, the more you'll hone your communication skills. And I'll say it one more time, communication is a critical skill for any senior engineer.

And speaking of communication, that's where content creation shines. As engineers, we love getting lost in technical details, but explaining complex concepts to non-technical stakeholders can be a challenge. By creating content, you'll learn to communicate your ideas in a way that's accessible and engaging to a broader audience. Plus, you'll have an opportunity to get feedback on your content and, if you listen to feedback, you can improve your communication skills even further.

Have heard of the concept of "atomic essays"? That just means breaking down your ideas into shorter, more focused pieces of content. This approach can make your content more engaging and easier to consume - especially on platforms like Twitter, where brevity is key. If you don't know what kind of content to create, this could be a good format to start with.

If you don't know what to create content about, here's my tip: every day, at the end of your work day write down 1 new thing that you think learned during that day. At the end of the work week, review your 5 points. I am sure that at least one of these will be worth creating some content about.

Creating content can help you become a more senior engineer by establishing yourself as a thought leader, developing your communication skills, and contributing to the community. So go ahead and share your knowledge with the world. You will be surprised at how much impact it will have on yourself and others!

## Other ideas

Here are some other random-ish tips that you can add to the ones above.

First off, always try to keep a positive attitude. When things get tough, it's easy to get discouraged and lose sight of the bigger picture. But remember, with enough time and resources, teams can overcome any challenge. When that luxury is missing, I am sure there are decent compromises that can meet everyone's needs.

Another key to upskilling is to avoid being too picky about technology or programming styles. Make an effort to expose yourself to different ways of solving problems. It's an

exercise for your problem-solving skills.

It's important to support our colleagues' ideas, even if we would have done things differently. This kind of collaborative attitude helps build trust and fosters a culture of innovation and teamwork. Also, next time you will propose something, I am sure that people will be willing to discuss and support your ideas.

When it comes to taking on tough jobs, don't be afraid to volunteer. What about that refactoring that nobody wants to do? Step up and take it on. Those messy, complicated parts of the code that everyone avoids? Dive in and try to make sense of them. By taking on these challenging tasks, you can develop a reputation as a problem solver and become the go-to person for the toughest jobs.

Finally, as a general rule, try to make things just a little bit better wherever you have the chance. Whether it's by documenting a process, improving an application's user interface, or streamlining a workflow, every small improvement counts. Over time, these small wins can add up and help us become more valuable team members.

---

## How do I sell myself as a senior?

If I think I am a senior engineer what can I do to be recognised as one?

First, start by taking on more responsibility. Look for opportunities to lead projects or mentor junior engineers. Share your knowledge and experience with others and provide guidance.

Secondly, try to make an impact beyond your immediate team or project. Participate in cross-functional initiatives, contribute to open-source projects, or speak at industry events. These types of activities can help you establish yourself as a thought leader in your field and demonstrate your expertise to a wider audience.

Thirdly, focus on developing your soft skills. As a senior engineer, you will be expected to communicate effectively with both technical and non-technical stakeholders, lead meetings, and negotiate effectively. So work on improving your communication, leadership, and conflict resolution skills to become a well-rounded and respected member of your team.

Another way to demonstrate your expertise is by sharing your knowledge with others. Write technical blog posts, record tutorial videos, or speak at internal or external events to showcase your expertise and demonstrate your willingness to help others learn.

Lastly, seek feedback from your colleagues, mentors, and managers. Ask for constructive criticism and be open to suggestions for improvement. By being receptive to feedback, you can continuously improve your skills and demonstrate your commitment to growing and learning as a senior engineer.

Photo by Miguel Bruna on Unsplash

## Conclusion

The definition of a senior software engineer is complex and multifaceted, but it generally involves possessing a deep technical expertise, a broad understanding of the industry, and a range of soft skills that enable effective communication, leadership, and problem-solving.

To become a senior software engineer, it's important to continually develop and refine your skills, focusing not just on technical abilities but also on communication, leadership, and business acumen.

This might involve pair programming, creating content, volunteering for challenging projects, and seeking out new learning opportunities.

Ultimately, the path to seniority is not a linear one, and it requires dedication, perseverance, and a growth mindset to continue improving and evolving as a software

engineer. So keep pushing yourself to go one level deeper, to have fun and find joy in your work, and to always strive to make a positive impact on your team and organization.

But if you have been wondering all the time whether I have been describing the *perfect* senior software engineer, let me only tell you that I am not expecting people to nail every single aspect mentioned in this article. There will be things you will be great at and others where you'll be barely sufficient. That's just human nature and it's okay!

We cannot excel at everything but we should know our strengths and weaknesses work with our team to amplify strengths and compensate for weaknesses ...and strive to get better every day!

It's a journey, not a destination! Touché...

**Found a typo or something that can be improved?**

In the spirit of Open Source, you can contribute to this article by submitting a PR on GitHub

## Comments



## Sections

## Share

## Similar posts

- 2022 - A year in Review
- 2021 - A year in review
- 2020 - A year in review
- 2019 - A year in review
- 2018 - A year in review

READ THIS NEXT

2022 - A year in Review

22 minutes read

LOIGE.CO

**EXPLORE**

Blog
Speaking
About
Comment Policy

**MY PROJECTS**

Node.js Design Patterns
AWS Bites
FullStack Bulletin
Middy

**FOLLOW ME**

Twitter
Twitch
Youtube
Linkedin
GitHub