Chapter 1

From Junior to Senior

This is a free chapter of **the Coding Career Handbook**! It is part of a sequence of **Career Guides** from Code Newbie to Senior Dev and Beyond, and it's yours to keep. Enjoy!

As you become comfortable as a Junior Developer (and, if your company has one, an intermediate level Developer/Software Engineer), you will naturally start looking toward the next level: **Senior Developer**.

What counts as a senior developer is not standardized, and everyone has strong opinions about it. To some, it is three years at a high growth startup. Others can take anywhere from two to eight years. Still others say they don't care about number of years (*and may or may not mean it*).

What other people think only counts so much. What really matters is what *your* company (or the other companies you interview at) looks for in a Senior, and whether they pay you commensurate with the market rate for Senior Developer. After all, a Senior title without the pay is meaningless!

If you're lucky, the company will have an Engineering Ladder where you can see their requirements for a Senior. If not, you can check **our discussion of Engineering Ladders in the Strategy section** (Chapter 26).

Ultimately, getting that role as a Senior Developer is a two step process:

1. Getting *enough* (not all) of the prerequisite skills and accomplish-

ments specified by the company

2. Successfully **Marketing Yourself** as meeting enough of those requirements to be hired into that role

This means that you often have to act like a Senior Engineer before you officially become one. Fortunately, this is *much* easier than the chicken-and-egg problem of getting your first job - most places will be happy to let you take on more responsibility while in your existing role!

1.1 Acting For the Job You Want

When surveyed, developers almost universally identify a few qualities that you should develop as you prepare for the next level:

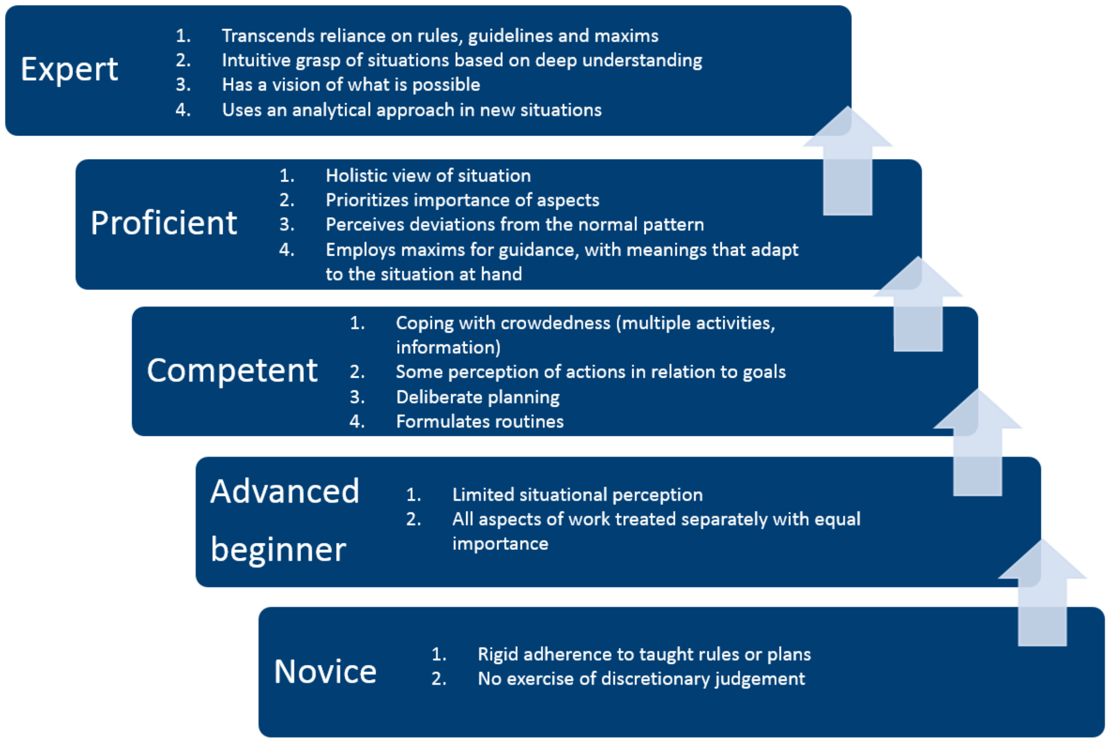
- Solid technical expertise, full command of fundamentals
- Impact on your team's work
- Being able to work across other teams
- Seeing the "Bigger Picture"
- Communication, Communication, Communication
- Mentoring others

Regardless of what your Engineering Ladder says, you will want to practice, practice, practice these skills as much as you can. They are just generally agreed upon qualities of a Senior Engineer that will help you and those around you throughout your career.

Note: More on this in the **Senior Dev** chapter!

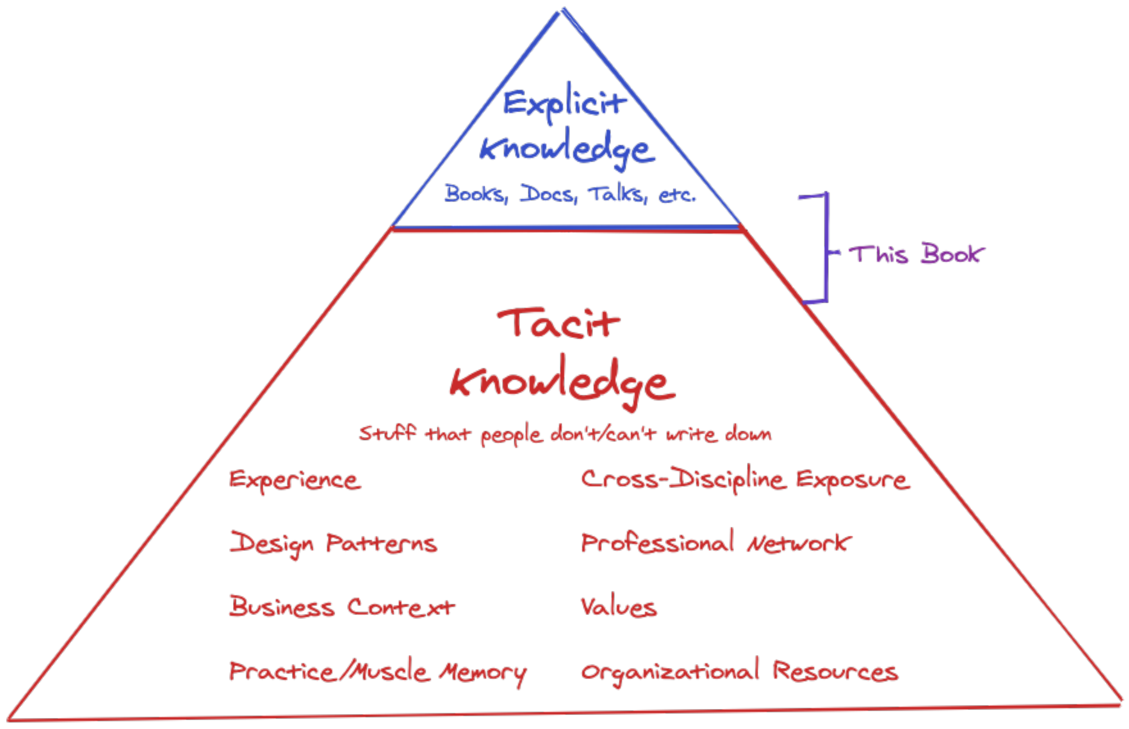
1.1.1 Technical Expertise

When it comes to your technical skills, consider how you are progressing along the Dreyfus Model of Skill Acquisition:


Expert
1. Transcends reliance on rules, guidelines and maxims
2. Intuitive grasp of situations based on deep understanding
3. Has a vision of what is possible
4. Uses an analytical approach in new situations
Proficient
1. Holistic view of situation
2. Prioritizes importance of aspects
3. Perceives deviations from the normal pattern
4. Employs maxims for guidance, with meanings that adapt to the situation at hand
Competent
1. Coping with crowdedness (multiple activities, information)
2. Some perception of actions in relation to goals
3. Deliberate planning
4. Formulates routines
Advanced beginner
1. Limited situational perception
2. All aspects of work treated separately with equal importance
Novice
1. Rigid adherence to taught rules or plans
2. No exercise of discretionary judgement

In your journey so far, you have likely progressed from Novice to Competent. As you go towards Expert in your field, you will likely want to pay attention to the meta-learning skills - focusing on **first principles** intuition (Chapter 17) when it comes to learning your trade.

Much of your learning from Junior to Senior involves gaining **tacit knowledge**. You can read all the programming books in the world, but, by definition, you are still limited to things that people can write down. That is **explicit knowledge**, and it is usually the tip of the iceberg when it comes to everything you need to know:



Tacit knowledge in engineering is a real thing. Keep a look out for all the lessons you don't learn in classes or from books. To *really* make your career explode, make a habit of writing them down for everyone else (Chapter 18 - **Write, A Lot**!).

You aren't alone in this journey – plenty of fellow developers have also written down their learnings. You can only get so far learning from your own experience – why not borrow the experiences of others? It's not the same as living through it yourself, but for example, reading through publicly published postmortems can teach you that many outages, even by the most well regarded companies, come down to error handling, configuration, hardware, lack of monitoring, and processes that allow human error.

1.1.2 Career Strategy

Many people define Senior Developers as "being able to see the **Bigger Picture**". Everyone agrees it's important, but nobody quite knows how to define it. It has something to do with how your technical work fits in context of the business/product, or fits in the broader architecture of the codebase, balancing both its history and future roadmap. So try to step back from your day-to-day work every so often and zoom out!

Before you make your big move, make sure you know what you want and take the time to position yourself accordingly in the preceding 6-12 months. Want to work on creating GraphQL APIs as a Senior? Better to do it as a Junior first. The logic here is: You aren't expected to make that much impact as a Junior, but you certainly will as a Senior. So it can be worth it to jockey around a little bit longer as a Junior or Intermediate Developer, just so you are in the perfect spot for a **career-making** Senior Developer role you can throw yourself wholeheartedly into. Better yet, your company can institute formal support for employees making internal "tours of duty", to grow you while keeping you!

It can help to make a list of what you've enjoyed in your current job. Among your peers (you *have* been building your network, right?), make a note of things that seem particularly exciting to you, and try to get exposure to those within your company or projects. It's a two way street - finding out what you like and are good at, and then positioning so that you can do even more of that.

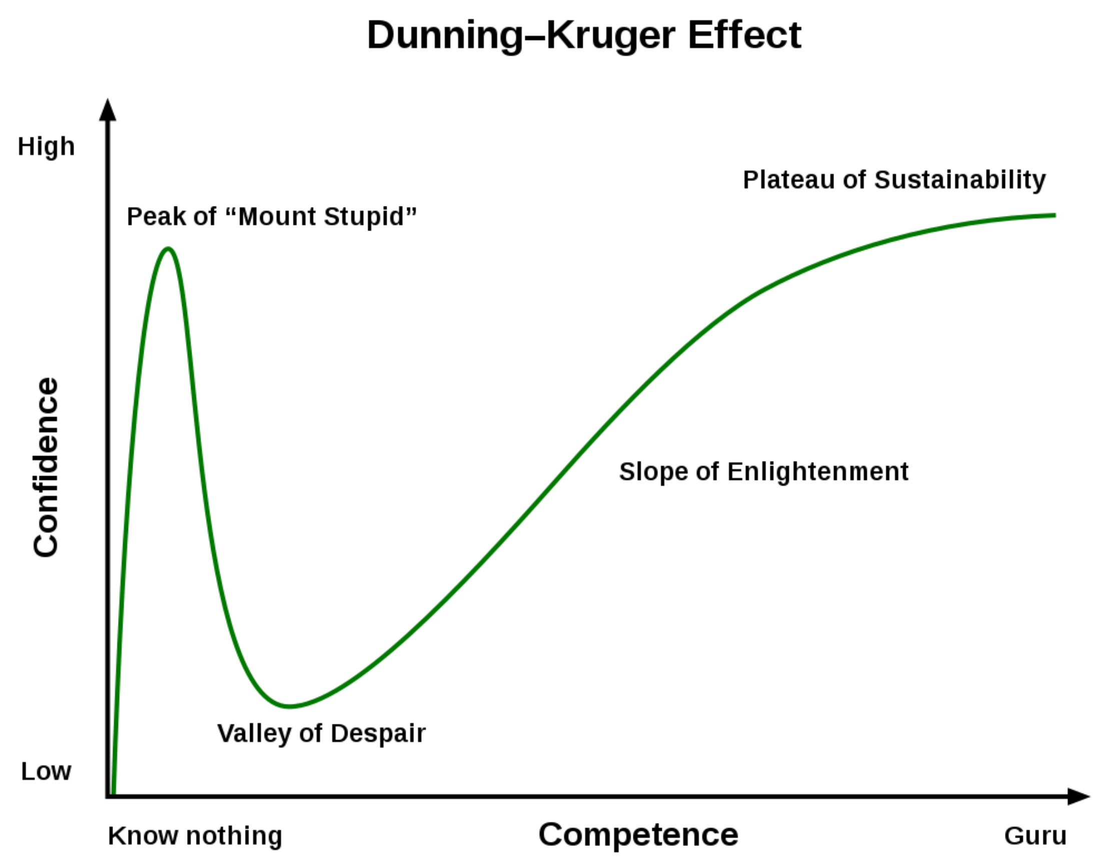
Note: This is the subject of the entire **Strategy** section of this book!

By the way - beyond just *what* you like to work on, you might also take note of *who* you like to work with. Here's an example from Keavy McMinn.

1.2 Marketing Yourself as a Senior Engineer

You may feel like you're not fully ready yet. You haven't checked off all the boxes for the Senior Engineer job or promotion you're applying for. **It doesn't hurt to try.** You don't know if it's impostor syndrome talking, and even if you fail the first time, you get *priceless* feedback and practice for the next time!

Remember the Dunning-Kruger effect between **what you know** and **what you know you don't yet know**:



People crossing from Junior to Senior are *particularly* likely to be at or just coming out of the "Valley of Despair". There's no point on that curve where you magically become irrefutably Senior. It's all a spectrum, and perhaps the only thing common among Seniors is having recovered from both the heights and troughs of confidence vs ability. You can ask other people how you're doing or teach what you know to keep some perspective.

Devs from minority backgrounds can face systematic bias (conscious or unconscious) in their evaluation. This is both unfair and a fact of life.

A sponsor with credibility can help you a long way – if you have trouble finding one, Mekka Okereke has a technique he calls "The Difficulty Anchor", which is hard work but a great strategy to win a powerful ally.

As you start marketing yourself as a Senior Dev, you'll want to have your accomplishments and stories in order. Just to give you an example, the AWS interview process involves asking you for examples of accomplishments and interactions that demonstrate one of their 14 Leadership Principles. Most companies may not be this formal about it, but will have some form of "Tell me a time when you…" question. Portfolios and **Proof of Work** still matter, but less so because you can choose to lean on a lot of the production work you contributed to as a Junior. Pay particular attention to any quantitative results you can cite – cost savings per year, Monthly Active User increases, Time To Interactive drops, whatever metrics make you look good.

In particular, anything you do in public - blogging, speaking, podcasting, making video tutorials, open source work - can help you grow your knowledge and your network at the same time, opening up the possibility for inbound opportunities to come to you. Remember to fight Impostor Syndrome every step of the way!

Note: You can find more ideas in the **Marketing Yourself** chapter (Chapter 39) of the Tactics section.

1.3 Junior Engineer, Senior Engineer

For Junior Engineers who want some ideas for directions to improve, it can be an interesting exercise to do a series of contrasting statements. I went through a long list of Junior-to-Senior advice online, and compiled these $\sim$30 comparisons in four categories: **Code, Learning, Behavior, Team**. *Please note that these are pithy opinions, not requirements!* We are all junior in some way, senior in others. You'll find that elements of these ideas permeate the Principles, Tactics, and Strategies throughout this book.

1.3.1 Code

- Juniors collect solutions. Seniors collect patterns.
- Juniors get code working. Seniors keep code working.
- Juniors deliver features. Seniors deliver outcomes.
- Juniors fix bugs *after* they create them. Seniors create tooling to *preclude* bugs.
- Juniors write tests because it's required. Seniors require writing *good* tests – because they've seen what happens when you don't.
- Juniors hate technical debt. Seniors have written code that *became* technical debt (and know when to let it be and when to migrate).
- Juniors love to keep code DRY. Seniors Avoid Hasty Abstractions.

- Juniors try to write the best code the first time. Seniors understand code is read, moved, copied and deleted far more than it is written.
- Juniors know how to use their tools. Seniors know when *not* to use them.

1.3.2 Learning

- Juniors make one-off side projects. Seniors use their side projects daily, often to make themselves more productive at their day job.
- Juniors learn to find the right answers. Seniors learn to ask the right questions.
- Juniors know what they need to know. Seniors know what they don't need to know.
- Juniors should absorb best practices from others. Seniors can derive best practices from first principles and personal pain.
- Juniors get stuck without docs and tutorials. Seniors aren't afraid to read specs and view source.
- Juniors might have strongly held beliefs. Seniors have had to *change* strongly held beliefs.
- Juniors question themselves when they fail. Seniors know they just need to give themselves more time and try again.
- Juniors stay on top of news. Seniors keep track of trends (especially **Megatrends** – Chapter 29).

- Juniors try to avoid mistakes. Seniors have made them all – and know how to recover.
- Juniors laugh at software tropes. Seniors know there's a grain of truth in all of them.

1.3.3 Behavior

- Juniors seek The Best. Seniors love the **Good Enough** (Chapter 16).
- Juniors should say "Yes" often. Seniors should say "No" more.
- Juniors should try to do the jobs they are given. Seniors should redesign their jobs as needed.
- Juniors complain about Open Source. Seniors understand Open Source only works thanks to contributors, not complainers.
- Juniors solve problems. Seniors identify problems before they become *problems*.
- Juniors start from what others say. Seniors start from what they need.
- Juniors know how to build. Seniors know when to buy.
- Juniors compare developer experience. Seniors look for hidden costs in user experience and in abstraction leaks.
- Juniors write as an afterthought. Seniors weigh writing as much as coding.

- Juniors leave comments. Seniors provide context.

1.3.4 Team

- Juniors work within their teams. Seniors know when and how to work across teams.
- Juniors grow their own output. Seniors grow their team's output.
- Juniors pair to learn best practices. Seniors pair to share expertise and see things in a new light.
- Juniors get roped in. Seniors get buy-in.
- Juniors must earn trust. Seniors *inspire* trust.
- Juniors seek out mentors. Seniors know how to learn from peers.
- Juniors work on improving themselves. Seniors work on improving their team, being a force multiplier through teaching, mentorship, and leadership.

Note that these are pithy, idealized comparisons just to get your imagination going on ways to improve yourself. In no way am I stating that any quality is unique to Juniors or Seniors, or that all Seniors or all Juniors practice all these qualities all the time.

1.4 To Stay or To Go

Finally, there's the question of whether to angle for promotion at your current company or to make the Senior Developer jump at a different company. That's a call you'll have to make, but **you will always be better off at least interviewing at other companies**. When you have an offer in hand from another company, you have a *pretty much airtight* case for promotion at your current company.

You've done the job hunt before; it'll be a lot easier this time. Beside hunting via the regular channels (online posting, networking at meetups and conferences), you should also be aware of new opportunities available to you at this stage. For example, recruiters of all stripes from in-house, third party, and venture capital will be more receptive to your cold emails. You can also tap your relationships formed online (via your writing or Twitter — you *have* been working on those, right?) to find opportunities before they get advertised. A warm intro of any sort beats applying via the front door and competing with everyone else on a 5 second glance of your resume. Finally, a big part of selling yourself as a Senior Developer is being able to communicate your level of experience in an interview - storytelling becomes a surprisingly big part of any senior hiring process.

Salary bumps are well known to be higher when you move companies - instead of a 5-10% bump, you could get a 50-400% bump because you could join a company with a different pay scale in a different industry at a different level in a different city. Of course these are major life changes, but higher bumps are more common when moving companies. There's also the simple fact that you didn't have much leverage when you were a junior dev. Now, you can actually take your time and practice some **Negotiation** (Chapter 31).

You may also wish to diversify your resume. If you move from junior to senior in the same company, you have less exposure to a variety of projects, technologies, opinions and cultures. If you're at an agency, you may wish to consider moving to a startup. If you're at a startup, you may wish to consider a BigCo. BigCo experience can net you big bucks at some forms of agency (including freelancing). So on and so forth. The earlier you are in your career, the easier it will be to hop around to figure out where you truly fit.

If you currently work at a place that doesn't have a good developer brand, you may wish to move to one that does, which will help boost your network and personal brand. Few people question the technical ability of ex-Google engineers.

Conversely, if you currently work at a place with a good brand, you may wish to take more risk in your next gig for more personal growth and financial upside, because the risk of failure is lower.