How to Go from Junior to Senior Programmer

A senior programmer is an expert who has made all the mistakes which can be made in his field.

Ravi Shankar Rajan
Mar 4 · 7 min read ★

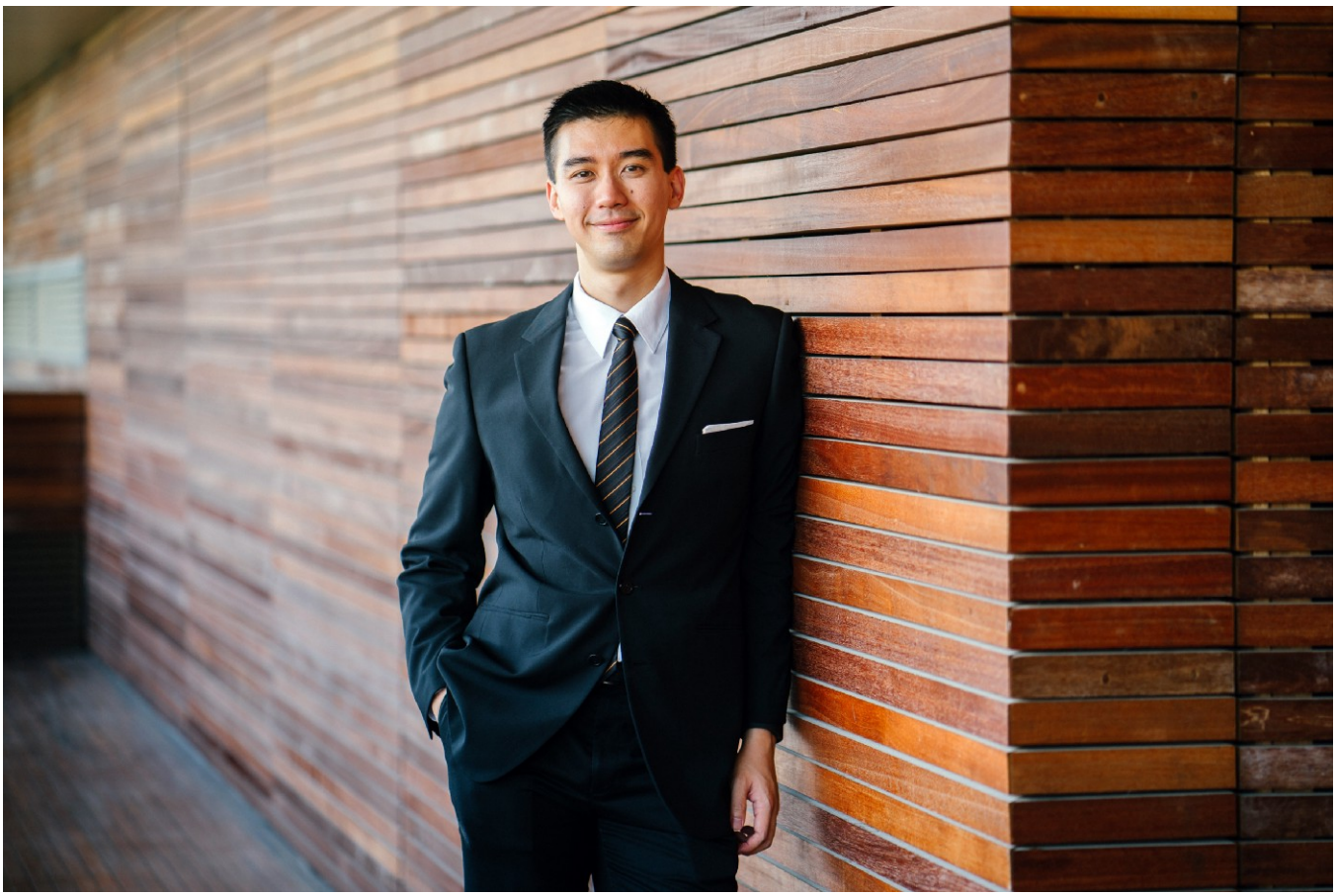

Image Credits: Pexels.com

A software developer classified by levels of experience can be something like this.

· Junior: 2–3 years of experience

· Senior: 10+ years of experience

· Medior or “mid-level”: in between junior and senior

This years-of-experience has a problem. It doesn't say anything about the quality of software development. How much experience and skill did you gain in those years? This is why job interviews for developers are so complicated. It's a hard to measure skill, so we end up giving difficult tests to developers during interviews. But these tests are at best, an approximation and cannot gauge the work or the expertise required while doing the job.

That brings us to the next question.

Once you're not a junior anymore, when do you become a senior?

Do years of experience automatically make you a senior? Not quite.

Take my case for example. When I was a young junior software programmer, I thought I knew it all. I was brash, cocky, and independent. I considered myself as the "God of code". I didn't like collaborating with others and I thought writing great code was the only thing that mattered.

I was wrong. Yes, coding is important. At the end of the day, a programmer has to write working code. But writing code is not the only thing that matters.

I learned this the hard way when I was working for my first customer. I spoke to the customer for "15" minutes, gathered the "gist" of the requirements and assumed that I know exactly what the customer wants. I started furiously working on the code like a madman, enjoying the act of writing pure code. I delivered the application in 3 days flat and then the customer rejected it. That was not what he was expecting.

Needless to say, I was hurt. My ego was bruised and I blamed the customer for not providing enough information. It was my immaturity speaking at that point in time. The customer never at fault. Had I devoted more time to analyze the customer's requirements, the situation could have been far different. I learned this the hard, hard way.

That said, a programmer isn't a programmer because he can code. He is a programmer because his job is to analyze everything before starting anything. The analysis has to be done at multiple levels.

· Self Analysis to perform better

· Analysis of customer requirements to deliver better

· Analysis of the overall project to help everyone perform better.

And if you want to go from a junior to a senior developer, you need to cultivate these analysis skills to metamorphose into a really good senior developer who is known for his expertise than the years of experience put in.

A good senior developer is like someone who's grown up, become an adult, and can take care of himself. His life is no more erratic, spontaneous and experimental. He has learned from his mistakes and has created a rock-solid foundation of expertise in his life, that he can look back and be proud of. He can be still "young", but what he has, is an abundance of pragmatism and effectiveness that will be far more valuable than his actual years of experience.

And here are some ways to go from a junior to a senior programmer.

. . .

Overcome the Dunning-Kruger effect

The Dunning-Kruger effect is a type of cognitive bias in which people believe that they are smarter and more capable than they really are. Essentially, low ability people do not possess the skills needed to recognize their own incompetence and this leads them to overestimate their own capabilities.

And as a junior programmer, this is a sure-shot recipe for disaster. You may assume that you are a rock star programmer and know everything but the reality is that you know very little and still far from achieving excellence. This is a trap you need to avoid falling in.

The difference between a junior programmer and a senior one is that a junior thinks he knows everything and a senior knows he has still a lot to learn. Junior programmers tend to overestimate their own knowledge and ability and are unable to recognize the skill and competence levels of other people, which is why they consistently view themselves as more capable, and more knowledgeable than others.

As David Dunning rightly states.

"In many cases, incompetence does not leave people disoriented, perplexed, or cautious. Instead, the incompetent are often blessed with an inappropriate confidence, buoyed by

something that feels to them like knowledge."

Dunning and Kruger suggest that as experience doing a job increases, overconfidence typically declines to more realistic levels. As programmers start deep diving in their areas, they start recognizing their own lack of knowledge and ability. And as they gain in knowledge, their expertise increases and their confidence levels begin to improve once again.

They suggest the following ways to overcome the overconfidence.

· **Keep learning and practicing** Once you gain greater knowledge of a topic, the more likely you are to recognize how much there is still to learn. This can combat the tendency to assume you're an expert, even if you're not.

· **Ask other people how you're doing**. Asking others for constructive criticism can provide valuable insights into how others perceive your abilities.

· **Question what you know**. Keep challenging your beliefs and expectations. Seek out information that challenges your ideas. The more you question, the more you learn.

Remember, feeling wise is pleasant. But you need to always raise your standards. And for that, you need to dig deeper in order to understand a particular topic better. It allows you to recognize how much there still is to learn.

. . .

Know when not to do something

In the book, The Subtle Art of Not Giving a Fuck, Mark Manson talks about the importance of maintaining an identity that is defined by as little as possible. That's because when we get our identities involved — when we decide that certain behaviors or events represent our worth as a human being.

In simple words, we often decide to do something based on how well it satisfies our ego or childish fascination for thrills rather than the real need to do the same. Manson tells us that the best decisions are made when we keep the "self" out of the decision because most likely, it's not about "you." Simply ask yourself, "Is this a good thing to do?" Yes? Then go do it.

This applies to programmers too. In fact, most programmers are magpies by nature, always collecting shiny things, storing them away and looking for connections. If you're not aware of this phenomenon, the shiny-toy syndrome is typically characterized by wanting to own the latest toy, often irrespective of the practical or functional need, or being hooked on the intense but very temporary high of ownership before moving onto something else.

If you are aiming to be a senior programmer, you need to avoid this disease at any cost. The best senior programmers know precisely when not to do something. They know that rewriting a library from scratch just to make it more readable, or switching to the newest framework from the older one are not always good decisions. The purpose of the code should be clear enough to grasp within minutes or even seconds. Navigating through the code should be easy, even without sophisticated wizardry.

The key is not to be risk-averse but just careful in picking the right battles.

. . .

Be insanely curious

Have you ever wondered what the word "application" means?

Why do we call those little icons on our smartphone applications? It's because they apply a given workflow or algorithm to a problem we have and help us to solve our needs.

That said, if you are building something, you are bound to make mistakes. Reflecting on your work and improving it continually leads to innovation and at the root of innovation lies curiosity to find out how things work. Remember it is an important clog in the whole cycle of self-improvement.

Mistakes->insights->curiosity->innovation->mistakes……. repeat……

If you want to move ahead and be a good senior programmer, you need to be insanely curious to get into everything you do. Curiosity is a tool that gets better the more you use it, and that's exactly what people expect in an excellent senior programmer. Good senior developers channelize their curiosity in a structured manner so they can use the information accumulated during crunch situations.

And here are some ways you can stroke your curiosity and build your brand.

· **Never stop learning**. Pick up a course, a book, an online degree, and use it to enrich the ideas you already have, and to get new ideas.

· **Focus on the fundamentals**. Make sure you understand how they work, so you can use it as the basis for your work.

· **Don't tell your ideas. Show your projects**. Ideas are overrated anyway. When your ideas are used and spread, you can build your brand.

· **Strike a balance between new and established ideas**. Don't blindly accept ideas about what you "should" know. Challenge the status quo.

· **Don't just make it work**. Make it extensible, reusable and pluggable. That is how you build your expertise.

Everything starts with curiosity. If you're not curious, you might as well quit. As Albert Einstein said:

> "I have no special talent. I am only passionately curious."

. . .

About the author:

Ravi Rajan is a global IT program manager based out of Mumbai, India. He is also an avid blogger, Haiku poetry writer, archaeology enthusiast, and history maniac. Connect with Ravi on LinkedIn, Medium and Twitter.