Alan (Zach Galifianakis) & Tyler (Grant Holmquist), The Hangover Part III, Warner Bros. Pictures

What it really means to be a "junior" developer.

Friday evening, I received an email from a buddy of mine who, right out of college (Rochester Institute of Technology), is working for a very promising and well funded startup doing C++ systems programming and machine learning. Below is a small chunk of his email.

One thing that bothers me at work dude is that although my co-workers are mostly all nice people, I feel as if I get the least respect for my work. I work with six engineers (making us a team of seven engineers). Of the six, one is Platform Architect, two are Senior Applications Engineers, and another is Software Architect (the other two are QA guys). Honestly, and I don't wan't this to sound arrogant, except for one of the Senior Applications Engineers, I actually find that I know a lot more than every one of these "senior" guys. Don't get me wrong…they all got many years doing this, working on some important systems and all, but I'm more knowledgeable than them. Most of the time just because I'm the Junior Systems Engineer, my ideas get thrown out the window and my hard work is not appreciated…in all honesty, that pisses me off to the max. Sometimes I feel like maybe I should go back to doing freelance (especially since I'm out of college now).

That email is 80% of the reason why I felt it necessary to write this post. 15% is from reading this, this, and this. The last 5% is from reading this.

I am a **junior developer**. My current job title may be just "Software Developer", but to the software industry, I am a junior developer. So what does that really mean? I have seen **too many** descriptions of what different folks feel is meaning of being a "junior" developer. I have also noticed that junior developers (including me, at some point in the past) associate a certain amount of **stigma** with the term. For the past few months, I have had the opportunity to work in a team of brilliant engineers building an awesome and rewarding product. Although just for about 6 months so far, having worked in an actual software team, I feel confident that I am now able to provide what I think is the appropriate meaning of a junior developer.

Rather than coming up with a formal definition, I will give some examples, from which you can derive the meaning. Okay…here we go.

Disclaimer for my buddy: I still love you bro. [:)].

The know-it-all.

I am willing to bet that 98% of us, junior developers, go through this phase at some point. I am going to use an imaginary character, Jack, as a way to better explain this.

As a young, enthusiastic, & passionate developer, Jack strives to be the best at his job. He is curious about everything, so he is always learning something new, whether it is a new programming language, paradigm, design pattern, or technology. He now knows seven programming languages. He knows how to write imperative, functional, event-driven, and object-oriented programs. Jack not only knows how to write fabulous factory methods, sexy singletons, delicious decorators, and prodigious prototypes, he knows when to properly use them (or at least he thinks he does).

Oh, and by the way, Jack also knows a ton about different environments, like Node.js, Java, Haxe, ooc, and even the one that you pushed to your private GitHub repository last night. Trust me, he knows it. OMG! How could I forget? Jack writes assembly too.

Because of all of this knowledge, Jack begins to develop, what I would like to call, a foolish sense of superiority. He now knows a ton more than average Joe, and therefore, he is better qualified for Joe's job. And please do yourself a favor, never call Jack a "junior" developer, or even worse, (uh oh, here it comes) a "cupcake"; if you do, please do not be surprised when you have a crazy headache later that night because Jack is sitting in his bedroom smacking a doll resembling you with a tire iron.

The experienced.

He who would learn to fly one day must first learn to walk and run and climb and dance; one cannot fly into flying.

Those are the amazing words of a great philosopher & poet, Friedrich Wilhelm Nietzsche.

See, there is a difference between having a ton of knowledge and being experienced. It took me a while to understand that, but the difference is quite interesting I must say.

Disclaimer: I am still learning and adapting to a lot of this stuff myself. I am **in no way** saying that these things do not apply to me as well.

Emotions.

Pospiech at the Polish language Wikipedia [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], from Wikimedia Commons

The more experienced developer knows how to break the less experienced in order to shape them. But…but…but WTH do you mean by that, Jonathan? It is really time you stop with the brown-nosing! I am already getting tired of it!

Okay, you really haz to calm down right now and listen.

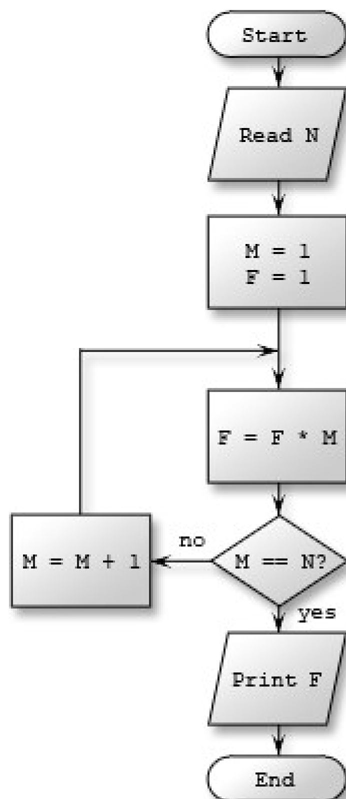
One thing that sits right next to young, enthusiastic, and passionate is what I have started to call “the staple”. You are said to have “the staple”, in my dictionary, if, as a developer, you are too attached to your technologies and productions. As the junior developer with these characteristics, you spend an extensive amount of time perfecting your code, thinking about all of the design patterns and principles that apply, writing unit tests (often really useless ones), *et cetera*. At some point, you finish and you are super duper confident about your work, so you go ahead and hand it all over to the “senior” developer, with a brilliant smile on your face of course. Then, to your surprise, she tells you that a design pattern that you used was unnecessary, your system is not horizontally scalable, and there is too much premature optimization going on. The little guy inside of you begins to cry, but you are strong, so you refuse to let her see it. You begin to wonder where dolls and tire irons are sold. She then goes on to explain herself. She tells you that, in a previous experience, the singleton pattern that you used, was used for that same purpose, where some basic dependency injection would have been a better fit, and thus left them (her previous team) swimming in a pool of refactoring. She then explains that writing to the server’s disk from your application, an application that is supposed to be load balanced, automatically disqualifies it as a good candidate for load balancing, because a file written to server 1's disk will not be readable on servers 2 and 3. She also tells you that the premature optimization of making all of your

variable names one character in length will only hurt you in the future as there are tools for that exact purpose. Hmm…"maybe I do not need that doll after all", you say.

That is what I mean by breaking you in order to shape you. Not only did you receive some expert criticism on your work, you learned a few new things along the way.

Good read: *The Number One Rule Of Programming Is Leave Emotion At The Door*.

Design and process.



By Cat, the Bob [Public domain], via Wikimedia Commons

Before being employed by Ai Squared, I was a freelancer. I had an appreciation for well architected products, and I had the foolish assumption that the stuff that I was building were well architected…they were **far** from that. So what is good architecture? That is one question that I am not yet qualified to answer, but, in a few years, I will come back to tell you. Okay? Deal.

One thing that I can say (so far) though is that there are two things that junior developers usually ignore that, in my humble opinion, are a must for engineering well architected products. These two things are, what I call, "design" and "process". We all know what "design" usually means, of course (hopefully), but what do I mean when I say "design"? Design is, simply, thinking about something that you want to build before you build it.

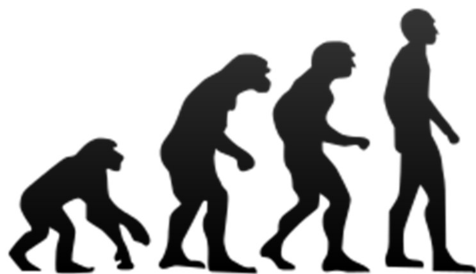
As junior developers, as backwards as it may sound, we have the habit of building before thinking and only thinking as we build. Being a freelancer, I was provided with tasks and left alone (well, mostly) to accomplish them. Because of this, I would crank out [poorly designed] code (minimum: 75 LOC/hour) for straight 15-20 hours…better than a goat (haha, I find it

funny that you did not know that goats code…#smh). All I had to make sure was that the client was happy and that their requirements were met. As long as the poorly designed code produced the output requested by the client, all was good and everyone was happy.

Okay, so what do you mean by "process" then? Process is a clearly defined route to be taken in order to accomplish a task. This is where project management, planning, and project management tools come into play. Again, before being actually in a software team, I knew about many project management tools, like Trello, Redmine, and Sprint.ly. In fact, I even used a couple of them, including Trello and Redmine. However, I had no process, so I treated these powerful project management tools like to-do lists (complete and incomplete type of thing). I knew, in theory, what QA was, but I did not really understand that process either.

I now have the opportunity to learn how to properly create processes.

The overall *déjà vu* proficient and the overall *jamais vu* expert.



By Tkgd2007 [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

There is nothing at work that I enjoy more than hearing my experienced co-workers describe mistakes that they have made in the past, the consequences of such mistakes, and what they learned. As a developer with an aspiration of reaching their level some day, it gives me the opportunity to skip a few mistakes that I would have otherwise made.

But Jonathan…your title! Okay, okay, allow me to explain.

The well-known expression "*déjà vu*" is a French expression literally meaning "already seen". Almost an antonym of this expression is "*jamais vu*", literally meaning "never seen".

Many junior developers (including me of course) know folks with "Senior" as a prefix or "Architect" as a suffix of their title whom we feel are less knowledgeable (in terms of programming-related skills) than us. One interesting characteristic of many of these folks though is that they have been around for a long time, worked for many companies (not necessarily), made many mistakes, learned from their mistakes, *et cetera*. However, unlike us, they might not know every language, environment, and/or technology out there. Instead what they do is they make themselves experts in a few areas (usually one or two, **as far as I have seen**), and instead of learning every language, environment, and/or technology out there, they pick maybe one or two languages, environments, and/or technologies, and make themselves **true** experts in those.

What does it mean to be a **true** expert in a language, environment, and/or technology? In my dictionary, a **true** expert in **anything**, is simply someone who is an expert, by not only knowing and understanding their domain at an expert level (inside and out, that is), but through years of experimentation and making use of such expertise. In other words, in my humble opinion, any developer with the time and commitment can become an expert in, say, a language like C# in maybe about a year (keep reading, keep reading), but to become a **true** expert like the honorable Jon Skeet, it takes **years**.

So what do you mean, Jonathan, by "overall *déjà vu* proficient". I am referring to the **true** experts. Their expertise may lie in only one or two areas, but they have "already seen" many things in those areas. They can build scalable systems, have a deep understanding of the ecosystem, know the shortcuts that lead to disaster, know when to optimize, *et cetera*. Woah! "scalable systems"…what is that thingy? Well, see, **fortunately**, at the moment, we (junior developers) fall under the "overall *jamais vu* expert" category. We may be experts (notice the omission of "**true**") in many different things, know how to build systems, have an understanding of the ecosystem, and know many shorcuts, but we (generally) have no idea how to scale the systems that we build, our understanding of the ecosystem is quite shallow, the shortcuts that we take are mostly never even properly evaluated, and we spend a lot of time over-engineering and optimizing prematurely.

Why the rush, folks?

By Fengalon [Public domain], via Wikimedia Commons

There is no need to rush.

If there is one article that I would never remove from my bookmarks, it has to be "*Teach Yourself Programming in Ten Years*" by the amazing Peter Norvig.

If you have not read it, go get yourself a cup of Mott's® Original 100% Apple Juice (cause it is good for you and of course since that is what I am drinking right now…[;p]), turn off the TV, Punch Your Brother in the Face™ (okay…I never said that), navigate to the article using the link above, and read it attentively.

Wait, I am puzzled…you never told me why being an “overall *jamais vu* expert” developer is **fortunate**. After all, my friend, this is the time of our lives when we can and are allowed to commit blunders, learn from them and others with experience. Using that to our advantage (in a good way of course) will only do us good.

Alright…I am done. The end.