Issue 034: Job Market

Graham Lee

July 5th, 2021

Welcome to the thirty-fourth issue of *De Programmatica Ipsum*, dedicated to the subject of the *Job Market*. In this edition:

- Adrian describes the painful recent evolution[1] of the Swiss tech job market.
- Graham reflects on the difficulties of the hiring process[2] and the obsolescence of the technical interview.
- In the Library section[3], Adrian talks about "My Job Went to India: 52 Ways to Save Your Job"[4] by Chad Fowler.

Enjoy this issue! Please subscribe to our free newsletter[5] to stay updated about new releases, share the articles on social media, or contribute[6] if you would like to support our work.

Cover photo by Andrew Neel[7] on Unsplash[8].

[1] https://deprogrammaticaipsum.com/corals-and-sharks/
[2] https://deprogrammaticaipsum.com/do-not-ask-me-about-how-interviewing-works/
[3] https://deprogrammaticaipsum.com/category/library/
[4] https://deprogrammaticaipsum.com/chad-fowler/
[5] https://deprogrammaticaipsum.com/newsletter/
[6] https://deprogrammaticaipsum.com/contribute/
[7] https://unsplash.com/@andrewtneel?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText
[8] https://unsplash.com/s/photos/junior-job?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Corals And Sharks

Adrian Kosmaczewski

July 5th, 2021

Until the mid-90s, the Swiss job market required two things of anyone interested in pursuing a career in management in any major local industry: a university degree... and a grade in the Swiss Army. As a consequence, in the Swiss side of my family I had a fair share of sergeants, lieutenants, captains, and other mid- to high ranking officers, most of them in artillery and infantry.

This was back in a time where there was no such thing as a job market, at least in Switzerland: the time of the Trente Glorieuses[1], the period roughly between the end of the Second World War and the 1973 oil crisis[2]. After entering a relatively important recession[3] in 1991, the Swiss unemployment rate, historically below 1% for almost half a century, hit an all-time high of 5.7% in 1997.

With unemployment so low, having a university degree and a grade in the military meant a simple thing: staying at the same company for 40 years, and then retiring confortably. The Swiss Dream, although we never called it that way.

The rationale behind such requirement was simple; if you could manage a group of soldiers, you were perfectly capable to manage a group of employees in any typical Swiss industry; banking, healthcare, watchmaking. Of course this was an absolutely false, but sadly not unexpected assumption in a country that did not allow women to vote[4] in federal questions until 1971, and in some cantons, until 1991.

[1] https://en.wikipedia.org/wiki/Trente_Glorieuses
[2] https://en.wikipedia.org/wiki/1973_oil_crisis
[3] https://en.wikipedia.org/wiki/Economy_of_Switzerland#20th_century
[4] https://en.wikipedia.org/wiki/Women's_suffrage_in_Switzerland

Needless to say, the (rude) awakening of the Swiss job market to the ideas of Agile, DevOps[5], and home office, still underway, will likely take ages to get a stronghold. This is by far the major factor holding back[6] the Swiss software industry as a whole.

Even though I did my compulsory military service in Switzerland in 1993, I did not earn a grade higher than "soldier", much to the dismay of my mother. I did get, however, enough verbal abuse, bullying, and even discrimination towards my family name for a whole lifetime. I also learnt how to use a radio system with analog cryptography that was already obsolete in 1965. Later on I dropped out of college, but that is another story[7]. For my mother, a boomer born in 1944, my chances to get a job vanished almost overnight.

Yet, almost 25 years after publishing my first website, and after 24 years of professional experience typing code in one way or another, I am still here. In the meantime I have migrated[8] from galaxy to galaxy as often as needed, and as a matter of fact, I can say that I am still a "marketable," specialized[9] expert, whatever that means.

Switzerland, the whole world actually, underwent (and is undergoing) a major transformation in the world of work. No, I am not talking about the inexorable transition to home office[10] due to the pandemic, but rather about the fact that, in the space of two generations, workers have been forced to evolve from corals to sharks.

I know, the whole "sharks must swim or die" is a bit of a myth[11], and also a bit of an overused cliché[12]; I am referring here to certain species of sharks that lack a swim bladder. These are the ones that do have to swim in order to avoid sinking to the depths of the ocean.

On the other side of the spectrum, corals live their lives in a reef, without needing to move much, filtering sea water for nutrients and providing a shelter to countless other species. They are fragile, and were already disappearing at alarming rates thirty years ago.

Corals and sharks. An apt analogy, applicable to our daily work as software workers, anywhere in the world. Painfully obvious to those working on any field; from databases, to frontend development, to mobile apps, to any programming language. The pace of innovation and change, needless to say, makes us swim constantly to avoid drowning and losing our "marketable" status.

We have all met corals in our work experience; there are still a few, most monitoring various legacy systems written in COBOL for OS/360 and still running in z/OS, Fortran, C++98, Python 2.x, Visual Basic 6.0, COM+, or even Ruby on Rails 1.0 applications.

Beyond the technological evolution, there is something else going on. For many years, young college graduates seemed to seek, through those classic whiteboard coding interviews[13], the free lunches and foosball tables common in FAANG[14], GAFAM[15], or Big Tech[16] companies. That, too, shall pass.

---

[5] https://deprogrammaticaipsum.com/issue-16-devops/
[6] https://akos.ma/blog/eight-steps-to-build-a-better-swiss-software-industry/
[7] https://deprogrammaticaipsum.com/teacher-leave-this-kid-alone/
[8] https://akos.ma/blog/the-developer-guide-to-migrate-across-galaxies/
[9] https://philipmorganconsulting.com/positioning-manual/
[10] https://deprogrammaticaipsum.com/issue-30-home-office/
[11] https://www.amnh.org/explore/news-blogs/education-posts/sharks-rays-myths
[12] https://www.workforce.com/news/office-politics-5-ways-to-survive-the-shark-infested-waters
[13] https://deprogrammaticaipsum.com/tales-of-the-interview/
[14] https://www.investopedia.com/terms/f/faang-stocks.asp
[15] https://www.investopedia.com/terms/g/gafam-stocks.asp
[16] https://en.wikipedia.org/wiki/Big_Tech

More and more software workers are launching their own startups, even in a conservative country like Switzerland. More and more software workers are aware and appalled by the steadily low numbers[17] of women and people of other ethnic groups than "white males in their 30s" in the industry. More and more software workers are concerned about the climate threat posed by cryptocurrency and planned obsolescence[18]. More and more software workers are becoming aware of the low ethics[19] standards sustained by Big and Medium Tech alike.

The next step for this new cast of sharks is to become the social species it needs to become, like the majority of actual[20] sharks swimming in the oceans in this planet, and unlike the solitary, gregarious hunter depicted by the media.

But there are other reasons for software workers to unionize. Many, as it turns out.

The computer software professional market still features higher demand than offer, even in spite of the incredible development of the past decades in education.

What many software developers do not realize is that in such conditions, "all other things being equal", computer worker salaries should go up, inexorably. This law is apparently (regrettably) only true in Silicon Valley. The lack of awareness of their shark status, and the avoidance to all sorts of unionization, mark the slow decline in salaries I have seen in Switzerland in the past 25 years.

Let me be clear in this point: what I mention here is a net decrease, *in nominal value*; I am not even factoring in the effects of the (admittedly low) inflation rates of Switzerland. There is a net decrease in the value paid to software workers in this industry in this country, and a net exponential increase in the gains of company owners, startup founders, and other members in higher positions in management. All of this at the detriment of actual workers, who have to face higher healthcare and overall living costs.

Ask around to older software workers in Switzerland, those who have been around for 20 years or more (and who have not quit in disgust, that is). It is an unspoken truth. This might be true in other markets, but I cannot tell. I would not be surprised if it is.

And now we start to see the raise of AI-powered coding systems, such as GitHub Copilot[21]. Currently in its first iteration, with somewhat laughable results[22], and a questionable[23] (to be polite) approach to licensing FOSS code. Still, GitHub Copilot marks the start of a new era. In short, the writing is on the wall: if you plan to make a living writing code in the next 40 years, I strongly suggest you start learning something else right now.

Michael "Rands" Lopp wrote[24] in 2004:

> Velocity. That's a better term than upward mobility. Constant forward momentum. How you are going to achieve your own personal velocity is your own deal.

In the meantime, until we, software workers, gather together and form worker unions to help protect our craft and defend our income, the only solution will be to behave like an

[17] https://deprogrammaticaipsum.com/issue-6-diversity-inclusion/
[18] https://deprogrammaticaipsum.com/the-twenty-year-computer/
[19] https://deprogrammaticaipsum.com/issue-5-ethics/
[20] https://en.wikipedia.org/wiki/Shark#Behavior
[21] https://copilot.github.com/
[22] https://twitter.com/di_codes/status/1410632313990488065
[23] https://twitter.com/iwasleeg/status/1410877095857827840
[24] https://randsinrepose.com/archives/what-to-do-when-youre-screwed/

intergalactic shark. Moving from technology galaxy to technology galaxy, and learning to trust our own negotiation skills to be able to grow in our careers, and eventually, to earn a bit more every year as an employee with ethics and dignity, without having to become a lieutenant or a captain in artillery.

Or even worse, to leave the industry altogether.

Cover photo by Kurt Cotoaga[25] on Unsplash[26].

[25]https://unsplash.com/@kydroon?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[26]https://unsplash.com/s/photos/coral-shark?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Do Not Ask Me About How Interviewing Works

Graham Lee

July 5th, 2021

It is very easy to get information about career progression in the software engineering world, but very hard to get good information about it. To understand how the various problems collude to stop us learning about our own career paths, let us start at the beginning.

I could tell you about my own experiences as a junior engineer (and as a candidate applying for those roles), but it is so long ago now that everything has changed. The technologies (to some extent, though in fact the popular languages of today are all the popular languages of back then) have changed. That first job I had as a sysadmin, which gave me the space I needed to learn about programming technologies and some problems to which to apply what I learned, I got by gaining access (physical or remote shell) to various computers by vendors who have long gone out of business and working out how to make them talk to one another.

But that is not all that has changed. The corporate cultures, and expectations, have changed too. I failed a Software Engineer in Test interview in Google back in my junior days because I did not have a Computer Science degree and they asked me a question which was a reification of the Traveling Salesman problem. They did not want me to try and whiteboard my way to a solution, like most interviewers would, they wanted me to already know that there was not a solution. I understand that Google have changed their expectations over the intervening decades.

The first company I was actually a software engineer for used a waterfall project management lifecycle, with projects across multiple teams organised into massive programs that were planned to take 100 people two years to complete (and invariably took at least 3.5). I may

have been paid for my Objective-C skills but I made my money by learning to exchange header files and UML diagrams with people called "systems engineers", whose exact responsibilities I was not certain of. Despite not understanding the system I could still survive within it.

And of course, I have changed a lot too. My recollections and reminiscences about those early days on the job are told through the rear view mirror of my later experiences, shaped by the person I became and my journey from there to here. The things I think are important now, I will tell you were important then, whether or not they were. For example, that first job where I got all the obsolete computers to talk to each other? I borrowed those computers from the guy who held the post before me, so I was learning to network in two different but both important ways.

Finally, we must realise that nobody on either end of the hiring process has much of a clue about what works. This is particularly true of hiring junior engineers, but hold that thought while we take a little look at why.

There are a few different examples of evidence you can gain about a candidate when you are reviewing their CV and interviewing them. There is credential evidence: what training do they have? Who gave that training? Is it up to date? What did it cover?

There is experiential evidence, which is quite broad. It overlaps a bit with credential evidence (if you rate Microsoft as a software company and this candidate has a couple of years of experience at Microsoft, that is effectively a certificate of Microsoft-worthiness). It also covers what they say they did, and what you can discover about what they actually did.

Then there is practical evidence, which a lot of software interviews focus on: how do you solve this problem? This can be done well (two interviews for different organisations that I took involved diagnosing actual customer support problems) or badly (reverse a linked list on a whiteboard).

And finally, the two more problematic ones: hypothetical evidence ("if you were asked to do a customer demo and the software crashed on launch, what would you do?") and personal evidence (there is a person sitting in front of you answering all your weird questions, what do you think of them?).

The problem you have interviewing a junior is that they likely have little in the way of experiential evidence: "have you ever X" is likely to be answered either with a straight no, or with a possibly-relevant story from some non-engineering part of their life (which does of course tell you something about how they handle abstractions).

The few people at the junior level who do have plenty of experiential evidence are the privileged ones who got access to the trammels of software engineering – computers, quiet time, documentation, permission – and could make it their hobby. Making use of this evidence is both entirely natural and a source of demographic bias in your hiring practices.

And the credential evidence they can supply is likely to be similar to all of their peer candidates: they have each just graduated or completed a coding academy, leaving nothing to pick them apart.

Interviewing, and thus hiring, at this level is a difficult and highly variable proposition. Therefore anything you learn about recruiting or being recruited as a junior engineer is heavily tainted by survivor bias: I got the job, I did this, ergo doing this lands you the job. Well, maybe. It has long been the case that there is both a healthy demand for, and supply of, junior candidates, so it is no surprise that the interviewing system for juniors is deeply

flawed. Weed out the dangerous candidates in interview, hire the questionable ones—then turn them into the employees you want or manage them out.

Does the situation get better as you go up the scale? Well, let us ask the other question: is there a scale to go up?

We have already looked at the management track, in our previous issue. It has significant flaws. But what about the technical track? Again, it is flawed, and again, the reason is survivor bias. Did you get that senior (Latin for "old", as in senile and senescence) role because you were fulfilling the junior role's expectations at a greater rate than other juniors? Or because you were doing bigger things than the other juniors? Or because there is an expectation of title inflation after a few years at your company, like a long service medal?

I will tell you how I got my first promotion. I was a software engineer at a company with around 100 Windows folks, and a Mac team of three people. I was the one with any experience of a Mac–both user and developer experience. Not very much developer experience, and I was not a very good developer, but still I was naturally the one person who could answer questions like "how do you do X" or "how much work is it to Y". De facto that made me the technical lead over the senior engineer with five years on the MS-DOS product and the new graduate: that coveted "senior" title was mine.

There are other ways to get it. If the company you work for is small enough you can be the CTO or President of Engineering just because they need their one coder to be the CTO whenever they go to a conference. But typically in larger organisations there is some length-of-service requirement: it is illegal to ask about years of experience (as a proxy for age discrimination, the one kind of discrimination that white men can find themselves on the receiving end of and thus the one that is taken seriously) so instead we make up a title progression and then ask what your job title is. We cannot distinguish ten years of experience from one year of experience, ten times over, but we are not trying to.

You may well be angry at this point. I have told you that my career path is contingent on my privilege, being the right sort of person, who knew the right other sort of people, in the right place, at the right time. And I have told you that there is nobody behind the curtain, no objective system that I have somehow gained: it is all privilege, network, and dumb luck.

Well, it is. You might hope that there is some technical component, like if you learn the correct programming language or cloud platform it will open up new opportunities. It will! It will open them up by getting you to user group meetings where you encounter people who want to hire in those technologies. Or by answering questions on the forums until someone asks you to come in on a contract. Not by providing you with technical skills that get objectively weighed up in an interview, that is complete fiction.

This tells you a little bit about how to view your career trajectory: as a social game in which you make your own luck until occasionally that luck pays off. But, more importantly, it tells interviewers and hiring managers what they can do to fix the broken software engineering job market.

For a start, ask whether you *really* need to hire more engineers. If you are hiring because your team cannot keep up with the defect rate, change your process before hiring more defect introducers. If you are hiring because your team cannot keep up with the feature request rate, fix your product managers before giving them permission to bump velocity to meet the "new capacity". If you are not hiring for any of those reasons, check whether your VP Engineering is mistakenly using headcount as a proxy metric for their own importance.

So you have decided you do want to hire an engineer. You can begin by hiring outside your network. You have already, by definition, got employees like that, you do not need any more (unless your company is a sweatshop/feature factory and you just need another interchangeable code typist).

Secondly, recognise that your hiring process is likely deeply flawed. Offer fair mutual flexibility, accept that any candidate's best position (for them and for you) may be outside the company, and support their growth whether that is through your career track or off into the wider world. If they leave with a positive experience of working with you, you have just grown your professional network, increased your attractiveness to other recruits, and helped someone along in their careers.

Photo by Hello I'm Nik[1] on Unsplash[2].

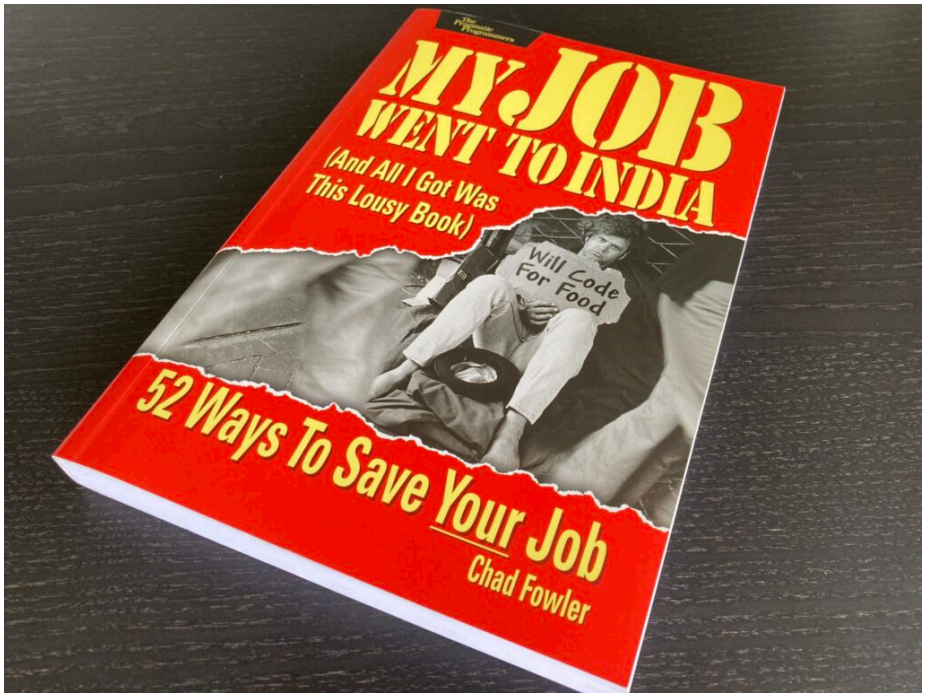
[1]https://unsplash.com/@helloimnik?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

[2]https://unsplash.com/s/photos/cv?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText

Chad Fowler

Adrian Kosmaczewski

July 5th, 2021



There was a time when I advertised my services as "Ruby on Rails" programmer. It was by that time that I got to learn the names and work of many people in that field; many of whom had come from the J2EE world, were tired of configuring everything in XML files, and preferred to use... YAML files instead. OK, I am being sarcastic here. Ruby on Rails was truly revolutionary when it appeared.

Chad Fowler[1] was one of the names that popped up constantly in the Rails Galaxy. A prolific author[2] between 2004 to 2013, musician and label[3] owner, he later moved to Berlin where he took care of the late Wunderlist application[4] until it was gobbled and discontinued by Microsoft.

Among his books, there are two that stand out, both flawlessly passing the test of time. The first being "My Job Went To India"[5], which is the subject of this article. The second one, "The Passionate Programmer"[6] deserves a De Programmatica Ipsum Library entry of its own.

When I read the former for the first time in 2005, the hype of the moment in the Swiss software industry was, without any doubt, offshoring. Companies of all sizes were setting up

[1] https://twitter.com/chadfowler
[2] https://www.amazon.com/Chad-Fowler/e/B001K8RU90
[3] https://mahakalamusic.com/
[4] https://en.wikipedia.org/wiki/Wunderlist
[5] https://www.amazon.com/Job-Went-India-Pragmatic-Programmers/dp/0976694018
[6] https://pragprog.com/titles/cfcar2/the-passionate-programmer-2nd-edition/

development teams elsewhere in the world; Romania, Morocco, Ukraine, Vietnam, Peru. For a little while it seemed to me that my job as a developer was going to disappear overnight.

The title of the book was (apparently[7]) a pun on an actual t-shirt sold by an American software engineer who had lost their job in 2004. But this is pure speculation. Said t-shirt was apparently made in sweatshops in Indonesia, a fact that sounded outrageous even at a time when there was not much awareness yet about the work conditions in those places.

Reading this book triggered me in various ways. I started marketing myself as an "Enterprise Architect" (whatever that means); I started working in technologies other than the ones provided by Microsoft (which had been my job up until that point); and I decided to study and get a Master's degree. Which I finished three years later. I also devoted more time to study Mac OS X and Cocoa, a choice that would eventually prove fruitful as I started a career as an iPhone app developer in 2008.

(So, Chad, if you ever read these lines; thanks a lot. Really.)

The book is a collection of useful tips and tricks to restart our careers in the face of offshoring and outsourcing. You can read it in a mere afternoon, but be ready to take notes. There is a lot of insight in it. I am not going to enumerate them here. Get your copy and read it.

But I digress; looking backwards, at least in the Swiss market, the promise of offshoring was only fulfilled to a certain degree. Companies still hired developers in the local market, and in staggering numbers. Code is still being written in Switzerland, in collaboration with engineers somewhere else in the planet, certainly, but a lot of code is written here everyday.

But for how long?

As I write these words, GitHub Copilot[8] has become the new sensation *du jour* in software engineering circles. Licensing issues[9] aside, it represents a wake-up call for all of us in this industry.

The current version of Copilot is a work in progress, available to a select few who applied to the beta program. The results are pouring in on Twitter, and even if many choices made by its AI engine are laughable at best, it is the firm conviction of the writer of these words that it will inevitably get better. Due to public outcry, the licensing issues will eventually be ironed out, and the engine will get better over time.

Depending on how you look at it, Copilot could represent the Sherlocking[10] of Stack Overflow. But this is a rushed appreciation. I think Copilot is actually more of the embrace, extend, and extinguish[11] strategy used by Microsoft.

And this time, the target is Free and Open Source code.

I wonder if there would ever be an updated edition of this book, called *"My Job Was Killed By Copilot"*. But I do not think this will happen. Maybe Copilot will fade away, and developers will just have to correct the code generated by it? Maybe Copilot will face the same fate as offshoring in Switzerland? Maybe.

To finish these thoughts, I must say that I would personally have added one more item to this book; "Unionize". You know that this magazine stands for and supports unionization

[7] https://www.theregister.com/2004/03/09/ny_times_in_my_job/
[8] https://copilot.github.com/
[9] https://twitter.com/bphogan/status/1411097686854488067
[10] https://www.howtogeek.com/297651/what-does-it-mean-when-a-company-sherlocks-an-app/
[11] https://en.wikipedia.org/wiki/Embrace%2C_extend%2C_and_extinguish

efforts in our industry, and this is becoming more pressing and urgent every day.

Cover photo by the author.