medium.com /@benbob/what-i-learned-from-working-for-both-bill-gates-and-steve-jobs-f0c04e1e5160

What I Learned Working for Both Bill Gates and Steve Jobs

Ben Fathi

15-19 minutes

> “When I was a boy of fourteen, my father was so ignorant I could hardly stand to have the old man around. But when I got to be twenty-one, I was astonished at how much he had learned in seven years.” — [Attributed to] Mark Twain.

Warning: I didn’t understand any of this as I was living through it. You won’t, either, when it happens to you.

It would be fair to say that I’m proud of my thirty five year career in the computer industry. I’ve had the pleasure of working with thousands of brilliant people and, thankfully, have learned a little bit from each of them. I was once a brash young engineer at NeXT Computer and, many years later, a Corporate Vice President at Microsoft. In a sense, two of the most important lessons I ever learned, I learned from Steve Jobs and Bill Gates.

I was a sophomore in college before I took my first computer science class, at the recommendation of an uncle who thought: “This computer stuff is gonna be big.” I was studying psychology at the time. I have no idea why. It was the best I could think of for a major when I entered college. I was set to graduate at seventeen, much too young to know what I wanted to do with my life.

I was breezing through college and completely bored with Psychology. It seemed like mental masturbation: just putting labels on people and on overlapping sets of vague symptoms. The fact that a single mental patient, when visited by five psychologists, will walk away with six diagnoses, is sufficient proof that psychology is more an art than a science. In the midst of all this, my first computer science class was a revelation.

*What? You mean there’s only one right answer to the problem? You mean the computer will do **exactly** what I tell it to do? And if the code doesn’t work, the problem is likely to be **my own damn fault**? Fuck, yeah! Sign me up!*

Here was a world that was much more satisfying than the vague world of psychology. So I did what every decent sixteen year old would do. I declared double major: Psychology *and* Computer Science.

“What the hell do those two topics have to do with each other?,” you may ask. Nothing really. I just happened to have already taken most of the courses needed for a bachelor’s degree in Psychology and wasn’t about to just give up on that! In the end, I still graduated at seventeen with both degrees and entered the workforce.

Funny enough, having now managed thousands of people and worked with tens of thousands of others, I find myself remembering many lessons from those Psychology classes. *Now* they make sense, now that I’ve seen dozens of examples of each symptom. Back then, I had no context. I hadn’t experienced enough of life to have a frame of reference. As such, the concepts seemed just like a bunch of empty words.

At the time I was a starving foreign student on an F-1 visa and, not having any immediate relatives in the country, my only path to permanent residency was to find an employer who would apply for a green card on my behalf. But there’s a catch. As a foreign student in the US, you can only work for a year after graduation on what is called “Practical Training.”

If you do a great job during that year, your employer applies for you, you get an H-1B visa which is then a path to a green card, citizenship, and the rest of the American Dream. If that doesn’t happen, you’re

out of luck and get to go back to your country of origin. Being of conscription age, I wasn't interested in going back to a country suffering through revolutionary turmoil and a pointless war. I would have ended up fighting in the front lines of the Iran-Iraq war. Thanks, but: No, thanks!

So I desperately needed a job and a sponsor. The only job I could find was at the local state university as a Computer Science lab manager. What a bizarre job for someone trying to break into the industry as a software developer. Well, that's the best I could do at the time. This is 1982 we're talking about, after all. Height of the Iran hostage mania, the oil crisis, recession, and all that.

I won't bore you with the details; it was not a pretty picture. Here I was, fresh out of classes programming the latest model PDP-11 and in love with Unix, having taken Artificial Intelligence classes writing code in LISP and Prolog, having studied heady theoretical automata theory… *and you want me to do what? Load these trays of punch cards into this 1960's era IBM card punch reader and change dishwasher sized disks on aging VAX systems? And this will get me a green card? Okay, I'm game. What the hell.*

It's only now, thirty five years later and a million miles away, that I'm actually thankful for having gotten to experience an entire generation of computing. One that was dying, for sure, but also one that allows me to contrast today's world even more starkly with where we were, just a few years ago.

Just think about it. Any kid can pick up a smartphone or tablet today, type in a question, *any* question, and get an instantaneous answer. You too lazy to type? No worries, just speak the words and we'll do the rest. Wow. Just fucking wow. Back in my day (can you hear the violin playing in the background?), we still had to go to the public library and use printed index cards to find reference books. If we were lucky, maybe the book would even be on the right shelf.

Do we even realize how far our world has come in just the past few decades? Fast forward fifty years, at the exponential rate we've been experiencing, and you'll see how far we'll go. I'm an optimist about the future if only because I've seen how fast this industry can move in the long run.

Don't get me wrong. In the short term, it's nothing but frustration and tedium, bureaucracy and cat fights, bug fixes and meetings. But in the long run, *oh my god!* Just take a big step back and look at how dramatically we have changed the human experience in just the past ten or twenty years.

I am a child of the sixties and seventies raised partly in a third world country. I still remember having to go to the national phone company office downtown with my parents and standing in line for an hour in order to make an international phone call. Today, anyone can connect with anyone else anywhere on the planet instantly through voice, video, email, and social media on a phone in their pocket. And they don't even need to drag an IBM card punch reader behind them or know Fortran to do so!

Holy crap! Now that's progress.

Of course, I didn't understand any of this at the time. I was just struggling to keep up with some of the best folks in the industry. It's only now that I see the consummation of all those things we worked on for so many years: the networking and security standards, the operating system platforms and ecosystems, the advances in usability and interoperability, reliability and scalability.

I still have trouble getting my iPhone to work with Google Play when I visit a friend's home (talk about vendor lock-in) but, once we agree on a platform, we can choose from thousands of movies, millions of songs, and dozens of shared experiences whether we're in the same room or halfway across the world. Twenty years ago, none of this existed. Now that's progress and we all had a hand in it. It's only when you take a giant step backwards and see the impact our industry has had, as a whole, on humanity that you can feel happy about your contributions.

Back to the story. There is no way a state university could legitimately apply for a green card for someone to be a lab manager, so I left after a short stint to find a better place. It took me three or four tries, at half baked startups and mediocre companies, to finally end up at a company where I could work on something I was passionate about: Operating Systems.

And I've never looked back. My entire 35 year career, before I finally retired last year, was spent working on operating systems, first as a Unix kernel developer and later as a manager, director, VP, and eventually CTO.

I spent a few years writing device drivers on Sun workstations, then did a lot of Unix kernel work at a multiprocessor high end server company. I got to work with every architecture from Motorola to MIPS to PowerPC, writing system components, device drivers, storage subsystems, virtual memory management systems, low level kernel code, doing system bring up, even soldering parts on the factory

floor when needed. Eventually, I made my way to the west coast and spent several years at MIPS and Silicon Graphics working on high end server systems. At its height, I worked on several supercomputer projects at Silicon Graphics.

When I tell people that, they immediately say: “Ah, Jurassic Park!” Well, yes. SGI was the company that built the graphics computers used for rendering many Hollywood movies, including Jurassic Park. But we also worked on supercomputers that competed directly with Cray Research for supremacy in the (then highly competitive) supercomputing world. Those were the heady days when I learned everything about computer architecture from the processor all the way up to operating systems and system software in general.

I seem to have worked on a lot of dead end system architectures. Supercomputers, UNIX workstations, shared memory multiprocessor architectures, RISC processors, tightly coupled server clusters: all architectures that have mostly fallen by the wayside as the world has embraced personal computing, the cloud, and distributed computing.

I used to fret about this. Why was I always killing myself on these herculean projects only to find out a few years later that a competitor had completely rethought the problem space and come up with a new generation of computing to address it?

It was only later that I realized: That’s probably true for almost everyone out there. Every architecture dies out sooner or later; that’s just the way this industry works. I’ve worked on many revolutionary projects — revolutionary when I was working on them — and every one of them has sooner or later been retired to the dustbin of history. Thankfully, with each generation we learn from the mistakes of the past. In the process, I also got an opportunity to work with some of the brightest minds in the industry and learn from them. The most important lessons took me years to learn.

Case in point: I was a brash young engineer who quit from NeXT Computer in 1992 when Steve Jobs cancelled the project I was working on: a PowerPC based next generation multiprocessor workstation running NeXTStep. The project was almost finished, the system was ready to be shipped, and was due to be announced at an industry conference the following week when it was cancelled. I was so mad that I didn’t even bother including the job experience on my resume!

Steve tried to get me to stay at the company but I was too hot headed and angry to realize that he had made the right call. He had realized that the battle over processor architectures was over and Intel had won. He completely killed all hardware projects at NeXT and gave the company a software-only focus. I, of course, stormed out the door. How dare he cancel *my* project?

I was too busy looking at the trees in front of me to see the forest all around. The processor war was over. The correct answer was to move up the stack and innovate in software, not to keep fighting the processor war for an ever-shrinking slice of the market. Of course, he then returned to Apple with the NeXT team intact and the rest is history.

That’s what I mean when I say the hardest lessons take years to internalize. I wasn’t thinking at that level. I was too emotional about the project I’d just spent so much time and effort on. I couldn’t be bothered to take a step back and look at the bigger picture. **What I learned from Steve later — much later, after I had cooled down — was to fight the *right* battles.** Continuing to fight a battle after the war has already been lost is an exercise in futility.

I was too stubborn to understand the lesson at the time but have internalized and used it many times since — mostly in scenarios that have nothing whatsoever to do with computer architecture.

> “We are never more (and sometimes less) than the co-authors of our own narratives.” — Alasdair MacIntyre, After Virtue: A Study in Moral Theory.

Later in my career, I spent a dozen years at Microsoft working on various versions of Windows. Now that you look back on it, you can see that Windows lost the mobile phone war to Apple, the server war to Linux, and the cloud war to Amazon. Back then, we were too busy pumping out versions of Windows to realize that.

It’s so hard to put into words the amount of organizational inertia that goes into an engineering team responsible for a successful platform being used by billions of people. They almost never see the disruption coming at them. Or if the leaders do, the rank and file don’t. Most of them are too busy pushing the current rock up the mountain, the classic definition of “Innovator’s dilemma.”

This is not a complaint about the leadership of Windows or of Microsoft. By the end, I was one of those “leaders”, eventually responsible for all core development in Windows 7 — arguably the most popular

version of Windows ever. I'm proud of what we accomplished as a team.

What I learned from Microsoft was how hard it is to build a successful platform that is used by billions of people, millions of apps, and thousands of companies. The more open you make it, the more programmable you make it, the more people that build solutions around and on top of it, the harder it is to innovate on that platform later.

What I learned from Bill Gates during those years was an amazing attention to detail. The man could sit through fourteen hours of non-stop meetings where one team after another would prance through, covering topics as divergent as operating systems, productivity apps, the internet, smart watches, video games, research, email, databases, browsers — you name it. He could drill down into the details with the best of them. Impressive mental capacity.

One of my favorite quotes is from the author Sam Harris: "Boredom is just lack of attention." But I prefer to turn the phrase around into a positive statement that best summarizes the lesson I learned from Bill: **If you pay enough attention, *everything* is interesting.**

I don't care if the topic is ancient Chinese ceramics or fluid dynamics, fantasy football or multiprocessor concurrency. It's all interesting if you just take the time to pay attention and learn its "language." You're not bored because the topic is boring. Chances are you're bored because you don't understand it. If you don't want to be bored, just pay attention. It's *all* interesting. I promise.

What I learned from Bill later, at a distance, was that he was also a decent human being. He could take that brain and apply it to solving much harder problems — education, poverty, disease.

I can sit here and write yarns about what I learned from each of the other smart people I've worked with over the years. That would take far more time than either you or I have and take up more pages than either of us would care to read — or write. More importantly, it won't mean much unless you experience it for yourself. Most lessons are lost on us until it's too late for them to have an impact. What I can tell you as a piece of career advice is to only work on things you are passionate about.

As long as you're learning, keep at it. There is so much to learn and this industry moves so quickly that you will fall behind if you stop learning even for an instant.

As long as you're moving in the right general direction, I used to tell people, it's all good. Don't try to plan out your entire road trip from New York to LA before you start out. If I'd done that, I would have lived an entirely different life — never having even signed up for that first computer science class.

Instead, on your way to LA, just make sure you're driving in a generally westerly direction, then keep going. And keep learning along the way, course correcting as necessary. You'll eventually end up in the right place; and you'll have a lot of fun along the way. I know I did.