12 valuable tips of a successful Developer at age 40

12 sharing from a man who became Developer after 6.776 days this effort will be the guideline for all who are fostering the dream of becoming a professional software developer.

About the author : The article was shared by Adrian Kosmaczewski - training specialist, software developer and a writing enthusiast, fluent in English, Spanish and French.

I am a 42-year-old developer (Developer) who is totally self-taught and this is the story I want to tell you.

A few weeks ago I read a tweet and it made me think about my career. The difficulties and challenges of the past few years suddenly flooded into my mind - where I began to take the first steps in my journey to become a Software Developer.

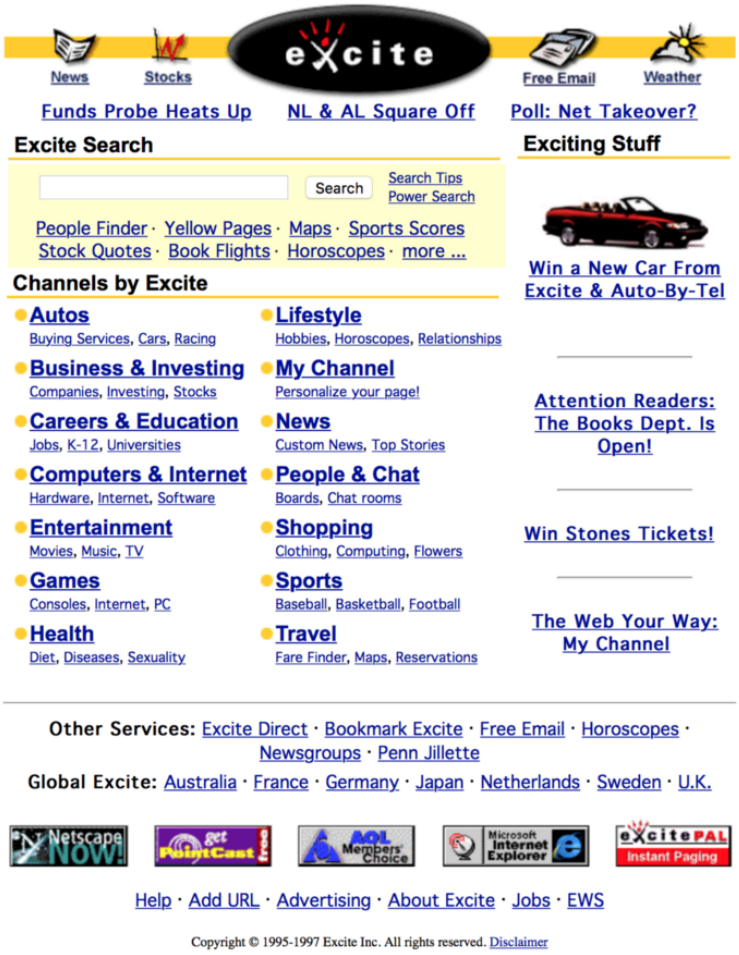
I started my career as an accurate Software Developer at 10 am, Monday, October 6, 1997, in an area in the city of Olivios, just north of Buenos Aires, Argentina. At that time, the value of "Unix's starting point" (Unix Epoch) was 876142800 (distance from 0 o'clock 0 minutes 0 seconds on January 1, 1970 to 10 o'clock 0 minutes 0 seconds on October 6 / 1997 is 876142800). Before long, I just celebrated my 24-year-old birthday.

World in 1997

At that time, the world was a little different.

Website does not have cookie warnings. The future of the web is portal, such as Excite.com. AltaVista is my favorite search engine. My email at that time was kosmacze@sc2a.unige.ch - which means that the personal website is located at http://sc2a.unige.ch/~kosmacze. Princess Diana died. Steve Job holds the position of CEO of Apple and convinces Microsoft to invest $ 150 million in the company.

Digital Equipment Corporation (DEC) at that time was suing DELL. Che Guevara's remains were found and taken to burial in Cuba. The fourth part of Friends comedy series began to be shown. Gianni Versace was murdered right in front of the porch. Mother Teresa, Roy Lichtenstein and Jeanne Calment (the people who lived the most in the world by that time) had just died. People play frantically on Final Fantasy 7 on their PlayStation. BBC 2 started radio programs through Teletubbies. James Cameron is about to debut Titanic. The Verve presented the hit version of Bitter Sweet Symphony and then paid 100% of the proceeds from that song for the Rolling Stones.



Smartphone (smartphone) at the time looked like **Nokia 9000 Communicator** , had 8MB in memory, CPU 24 MHz i386 and ran GEOS operating system.

Smartwatch (smart watch) at the time looked like the **CASIO G-SHOCK DW-9100B** , there are not many applications like today but much longer battery life.

IBM Deep Blue first defeated chess champion **Garry Kasparov** .

A hacker named *"_eci"* has published a C code that creates security vulnerabilities for Windows 3.1, 95 and Windows NT called WinNuke - a DoS (denial of service attack) attack on TCP at port 139 (NetBIOS) causes Blue Screen of Death.

By chance, 1997 was also the year that Malala Yousafzai, Chloë Grace Moretz and Kylie Jenner were born.

The plot of many films was also completed in 1997, such as Escape from New York, Predator 2, The Curious Case of Benjamin Button, and Harry Potter, Harry Potter. and the Half-Blood Prince (Harry Potter and Prince Lai), The Godfather III (Godfather 3) and the Terminator 2: Judgment Day, Skynet will begin to realize itself. at 2:14 am on August 29, 1997. This did not happen; However, another interesting thing that happened was that the **Google.com** domain was registered on September 15 of that year.

Until 1997, two years after the Y2K event and the media started making people worry about it.

Something like, at the time when I started dreaming to become Developer, the world was like that.

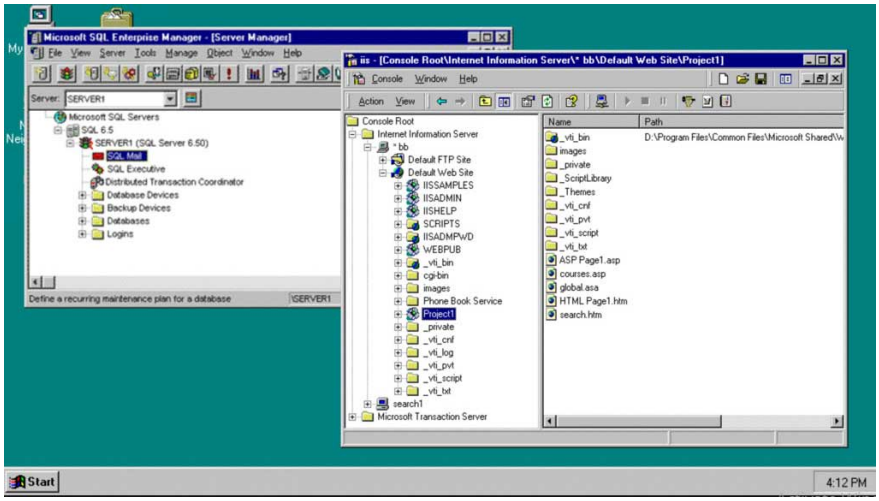
My first development work

My first job included writing ASP pages at several publishers, from **Microsoft FrontPage** , **HotMeTaL Pro** to **EditPlus** , managing multiple browser compatibility between **Netscape Navigator** and **Internet Explorer 4** , writing **stored procedures.** (a set of SQL statements used to execute a certain task) in SQL Server 6.5 provides an e-commerce website that operates in Japan, Russia, Britain and Spain - without Any support of UTF-8 (a very popular encoding method for describing Unicode encoding on memory) via software stack.

The product generated from these efforts runs on Pentium 2 servers located somewhere in the US with up to 2GB of hard drive and 256 MB of RAM. It is the only server running Windows NT 4, SQL Server 6.5 and IIS 2.0 serving about 10 thousand guests per day.

The first programming language I used was called **VBScript** and it certainly has a bit of client-side JavaScript - sometimes with lots of commands like *"if this is Netscape do this, else do that"* (If it's Netscape) then do this, if not then do that) because at that time, I still didn't understand how to use JavaScript correctly.

What's interesting is that it's now 2016 and we almost don't understand how to do anything in JavaScript.



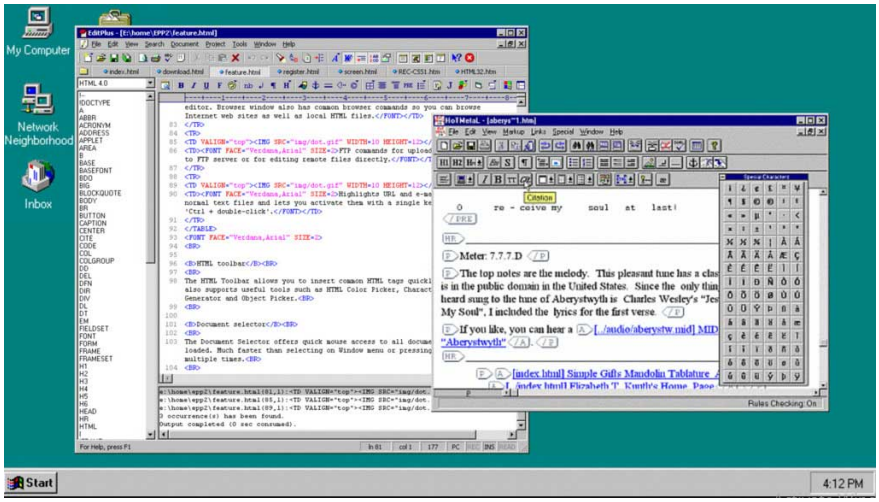
Unit test (a technique to test the operation of all code details with a process of separating software development processes to detect flaws .) has never been heard. **Agile Manifesto** (**Agile Manifesto** - flexible software development) has never been written. **Continuous integration is** a dream only. XML is not even a common term. The QA strategy only includes rebooting the server once a week because if you don't do so, it will stop working at all times. We developed **COM + Component** in Visual J ++ to parse JPEG files that have been uploaded to the server. As soon as the encrypted **JPEG 2000** files begin to appear, our Component is also horribly destroyed.

We do not use **source control** , including CVS, RCS or SourceSafe. Subversion doesn't exist yet. Our Joel Test score is -25.

6,776 days

In 6,776 days ago, I drank a cup of coffee every morning and wrote code in languages ??like **VBScript, JavaScript, Linux, SQL, HTML, Makefiles, Node.js, CSS, XML, .NET, YAML, Podfiles, JSON , Markdown, PHP, Windows, Doxygen, C #, Visual Basic, Visual Basic .NET, Java, Socket.io, Ruby, unit test, Python, shell scripts, C ++, Objective-C, batch files** and most recently **Swift.**

In 6.776 this day, a lot of things happened; Most importantly, I am married. I jumped 6 times and was fired 2 times. I started my own business and soon closed it. I completed the Master's degree program. I have released several open source projects and one of them helped me appear in an article published in Ars Technica (author Erica Sadun). I also participated in the **Swiss and Bolivian** television program. I watch live presentations by Bill Gates and Steve Jobs in Seattle and San Francisco. I also participated in speeches and co-hosted conferences on 4 continents. I wrote and published 2 books. I was exhausted twice (not because of books, myself) and many other things happened, both wonderful and bad things.



I often think about leaving this job. However, somehow, those lines of code always remind me of the past. I like to write applications, systems, software. To avoid exhaustion, I have to develop tactics.

In this article, I will share with you my 12 experiences. You can consult or choose to apply for yourself. But anyway, I still believe that 40 years old, you can still become a Developer if you are willing to pursue it.

12 tips for new developers

1. Forget the exaggerated words

My first advice for all of you is to ignore the exaggerated words. Each year, there will be a programming language, framework, library, pattern, component architecture or emerging paradigm that makes bloggers suddenly achieve resounding success. People are crazy about it. Workshops are organized. The book is written. The Gartner hype cycles chart increased and fell continuously. Consultants receive huge amounts of remuneration to teach, deploy . or "destroy" the lives of many people in this area. Newspapers support those chaos and will make you feel guilty if you ignore them.

1. In 1997, it was CORBA and RUP.
2. In 2000, it was SOAP and XML.
3. In 2003, it was a Model Driven Architecture and Software Factory (a structured collection of related software assets).
4. In 2006, it was Software Factories and OLPC.
5. In 2009, it was Augmented Reality.
6. In 2012, it was Big Data.
7. 2015 . virtual reality or Bot?

Don't worry about exaggerated words. Keep doing what you are doing, learning what you are learning and moving forward. Only consider them if you really care or if you feel that they can give you short-term or long-term benefits.

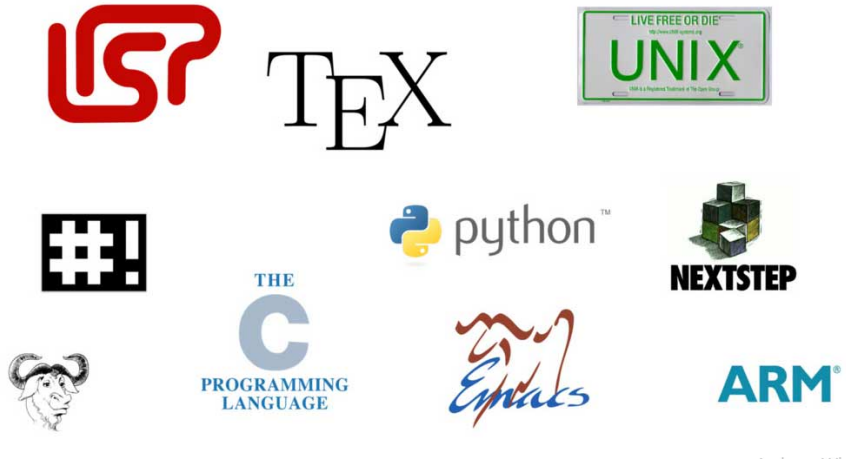
The reason for these lies is based on the fact that, like the Romans once said, *"nihil sub sole novum"* (nothing new in the sunlight). Most of what you see and learn about computer science has appeared or appeared in the last few decades. This fact is intentionally covered under a variety of marketing theories, books, blogs and questions on Stack Overflow. Each new architecture is merely a re-imagining and re-emulating an idea that has been

"floating" for decades before.

2. Select "galaxy" wisely

In this area, each technology will create something that I temporarily call "galaxy" (galaxy). These galaxies stand out for stars but also black holes (black holes); meteor stars fly across the sky every night, many planets but only a tiny fraction of those planets exist in life, cosmic dust and dark matter.

.NET, Cocoa, Node.js, PHP, Emacs, SAP . are galaxies. Each galaxy stands out with missionaries, developers, bloggers, podcasts, seminars, books, training courses, consulting services and other related issues. Galaxies are built with the assumption that their underlying technology is the answer to all problems. Therefore, each technology is based on a false hypothesis.



Developers from different galaxies embody "prototype" views that will bring that technology to life. They stick with ideas and will enthusiastically wear T-shirts, and convey to others about their benefits from that choice.

Indeed, I use the term "galaxy" to avoid having to use another term that might cause irrelevant arguments "religion" (a religion) - words that can be described more correctly. phenomenon that I want to say.

Personally, I spent the first 10 years of my career in ***"Microsoft galaxy"*** and 9 years later devoted to ***"Apple galaxy".***

I dare say, one of the biggest reasons why I change those galaxies is Steve Ballmer (former Microsoft CEO). I was too tired by the general attitude of those in the *"Microsoft galaxy"* against open source software.

On the other hand, I must say that the *"Apple galaxy"* is a great place, full of excellent artists, musicians and writers - those who, by chance or not, are also interested in establishing software process.

I also attended *"Microsoft Galaxy"* conferences, such as Barcelona TechEd in 2003 or Tech Talks in Buenos Aires, Geneva or London. I was even a speaker at Microsoft DevDays (Geneva) in 2006. Most of the developers in the " unfriendly " , *"unity"* and *"Microsoft galaxy"* are secretly and rife with each other. Non-disclosure agreements (NDA) and cumbersome IT processes.

Back in 2006, the *"Apple galaxy"* for me is the exact opposite: full of people who are musicians, artists, artists; those who write software to satisfy their passion and they also write with that passion. It made all the difference and that day, I still enjoyed this "galaxy", the "galaxy" that we all were in, right now and it connected us. together.

At that time, iPhone was born and the rest belonged to history.

So, my proposal for you is: **choose the galaxy wisely, live your life as best or superficially as you want, but keep your telescope toward other and available galaxies. Ready to jump far into a new place if necessary.**

3. Learn about software history

Learn how your favorite technology appears. Do you like C #? Do you know who created it? How is the .NET project formed? Who is the leading architect? What are the obstacles of a project and why is language becoming something as important as it is today?

Apply this formula to any CPU language or architecture you like or love: Python, Ruby, Java or any other programming language: learn about their origin, they appear like that. Come on. Similarly, continue to do so with operating systems, networking technologies, hardware, and everything else. Discover and learn how people approach these ideas, how long it takes them to master and conquer them. You know, a good software also takes 10 years to complete the development process.

Stories around the origins of the IT industry are fascinating and show you two things: *first* , everything is a disturbance again. *Secondly* , it is because of the first that you can become the one who shuffles the next big thing. No, change your mindset like this: **you will become the creator of something next great.**

And to help you achieve that, I'll give you some historical titles that I really like:

1. Michael A. Hiltzik's Dealers of Lightning.
2. Revolution in the Valley by Andy Hertzfeld.
3. Eric S. Raymond's Cathedral and the Bazaar.
4. The Success of Open Source by Steven Weber.
5. The Old New Thing by Raymond Chen.
6. The Mythical Man Month by Frederick P. Brooks Jr.

You will also learn how to appreciate things that still stand before time shift: **Lisp, TeX, Unix, bash, C, Cocoa, Emacs, Vim, Python, ARM, GNU make, man pages.** These are typical examples of technologies that benefit in the long run - things that are welcome, praised and learned.

4. Continue learning

Learn. Learn anything. Want to learn Fortran? Let's start. Feeling Erlang interesting? Great. Thinking COBOL could be the next big thing in your career? Great. Need to know more about Functional Reactive Programming? Okay! Design? Surely the better. UX? You must learn. Poetry? Should learn a bit.

Many popular concepts in Computer Science that have been around for a few decades have made learning about old programming and framework languages ??even more valuable, even if they are "secret".*First* , it makes you appreciate the current state of the industry you are doing (or hate it) and *second*, you will learn how to use existing tools more effectively - probably because you will understand its origins and heritage.

Tip 1: Learn at least one new programming language each year. I am not the one who proposed this idea but this book: The Pragmatic Programmer. Of course, it's effective.

A new programming language every year is not too difficult. Step into the *"hello, world" stage* and create something useful using that language. I often build a simple calculation software with whatever new technology I learned. It helps me to clarify the syntax and get familiar with API or IDE .

Tip 2: Read at least 6 books per year. I have listed an above list of 6 books that must be read that can keep you busy for about 1 year and below are 7 books for the following year that you can refer to:

1. Peopleware of Tom DeMarco and Tim Lister.
2. The Psychology of Software Programming by Gerald M. Weinberg.
3. Robert L. Glass's Facts and Fallacies of Software Engineering.
4. Don Norman's Design of Everyday Things.
5. Agile !: Bertrand Meyer's The Good, the Hype and the Ugly.
6. Rework of Jason Fried and David Heinemeier Hansson.
7. Geekonomics of David Rice.

6 books a year may sound like a lot, but it takes **2 months to read only one book** . And most of the books I mentioned above are not too long, easy to understand, funny and contain lots of interesting things.

Look at the problem in this direction: **If you are now 2 years old, by the age of 30 you will be able to read more than 60 books and 120 books when you're at my current age. And at that time, you were also "kidding" with at least 20 different programming languages.** Just thinking about this for a second was happy enough to want to start right away!

Some of the 12 books I have chosen for you above were written in the 17th century, some in the 18th, 19th and last centuries, mostly written about 10 years ago. They are great pieces of software development that I have read.

However, don't just read them. Take notes, mark, write on those pages and then take time to read them again. Borges once said that **satisfaction is greater than reading a book and re-reading it** . And of course, please buy paper books if you really like them. Trust me. E-books are overestimated (compared to their reality). Nothing can defeat things that are really grasped.

Obviously, you also need to know that when you start getting old, the number of things that are supposed to be new and / or important will start to drop very sharply. Be prepared for that. It is not surprising if you feel sad when you recognize it.

5. Teach others

Once learned, teach others. This is very important.

This does not mean that you should rent a classroom and invite people to listen to you "babbling" (although it would be better to do so!). What I mean is that for questions posed on Stack Overflow, give the best and meaningful answers; write a book; publish a podcast to share your favorite technology; write a blog; write on **Medium** page; go to other countries, open class / school programming by using the Raspberry Pis (a microcomputer) or help new developers enter the industry by becoming their adviser (though , don't do this before 30 years old because you're not experienced enough.

Returning to others will help you be more humble because this process will show just how narrow your knowledge is. This is also a great way to learn. Only by testing your knowledge with others will you learn the right way. In addition, communicating to others what you know will also make you more respected by developers and other technology enthusiasts; Each language, regardless of its trivial or "noble" status, has its place in the Tao of Programming and only through teaching can you feel it.

And during the process of teaching others, you can really, really make a difference in this world. Back in 2012, I received an email from someone who attended one of my training sessions. She used to work as an Adobe Flash developer. Do you remember ActionScript and all that? Not surprisingly, after 12 years working as a freelancer in Flash, she suddenly felt unemployed, lonely with a child to take care of. In the email she said that when she attended my training, she was very excited and also learned some useful things. Later, she found a job in mobile web development. She sent me a thank you.

I dare not claim that I can change the world but perhaps, I can make it "shift" a little to another position (hopefully) better. This thought made the lesson that I learned from that time become much more valuable and

meaningful.

6. The working environment is terrible

Don't expect software companies to open a career path for you. If in the US, maybe this will happen but I have never seen them in Europe. This means **you are forced to take responsibility for success in your career.** No one told you, *"Oh, yes, next year you can develop to become a leader, then the manager, then the chief technology officer ."*.

There is no such thing. The reality is the opposite: you used to be, you are and will be a software developer, that is, a factory worker who has relatively "price", the jobs of your manager completely can use outsource services regardless of what they say to you.

Don't take a job just for money. Software companies are slowly becoming sweaty factories - where you are supposed to find a way to prove your unreasonable high wages by "unknown" hours and expectations. exceeding. And, in the case of Switzerland, there is no union that will help you if things get worse. Indeed, there is a union in this country too, but they don't really care about situations that aren't going to create a real media crisis.

Any time the HR manager says *"you have to do it because we pay you"* , remember the following sentence to answer: *"You pay me but I have given you the gray matter My and I refuse this "deal".*

And the worst thing is that they will put you in an open space and for some reason they will be proud of it. Open spaces can be a ulcer. They do so without any doubt that they will create an office that is extremely lousy and unsuitable for software development - or whether a project needs brain activity. .

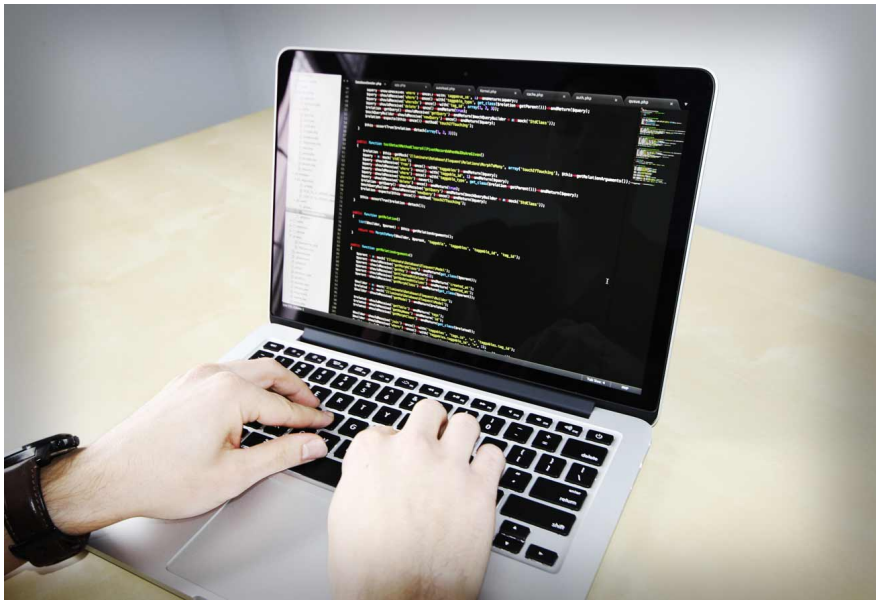
Remember this: the **fact that you understand something doesn't mean you have to agree with it.**

Outside there are good working environments, not many but certainly there are. I am very lucky to work at such places. You can also. Don't let the tedious work kill your enthusiasm. It is not worthy. Break the law and move.

Or better yet, become an independent developer.

7. Understand your value

You may have heard the mystery about *"10x Software Engineer"*. In fact, this is no mystery, but it does not appear in the way you still think it will.



"10x Software Engineer" is the person who makes 10 times the value no matter how much the employer "pays" them. They always make things with values ??many times greater than what the boss thinks they can do. And sure enough, they will receive bonuses and even shares. Read **Karl Marx** and **Thomas Piketty** to know your value.

Be like a shark that keeps swimming because your skills are extremely valuable. Dare to suggest your desired salary based on the ability for bosses and colleagues to understand what you can do. B?n càng d?ng c?m công khai mong mu?n c?a mình thì s? b?t công s? ???c làm d?u.

8. Hãy bi?t khiêm nh??ng

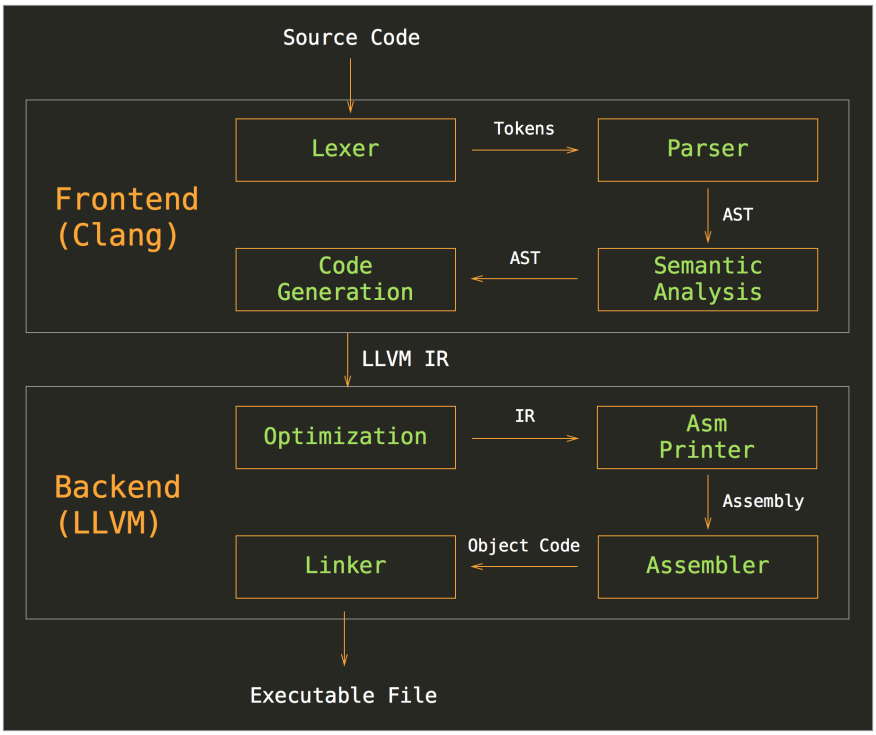
N?u b?n là ng??i da tr?ng thì hãy nh? m?t ?i?u r?ng t?t c? nh?ng ??c quy?n mà b?n ???c h??ng t? lúc sinh ra ch? b?i vì b?n ???c sinh ra theo cách ?ó. Trách nhi?m c?a b?n là thay ??i cu?c s?ng c?a b?n và xóa b? ??nh ki?n r?ng b?n ?ang x?ng ?áng ???c h??ng nhi?u ??c ân h?n ng??i khác.

Hãy ??a ra các quy?t ??nh d?a trên nh?n th?c c?a b?n. Hãy nh?n ra nh?ng hành ??ng và tác ??ng c?a chúng t?i m ?i ng??i xung quanh. ??ng e th?n hay c?m th?y ng??ng ngùng khi thay ??i ý ki?n. Hãy nói *"tôi xin l?i"* khi ???c yêu c?u. Hãy l?ng nghe. ??ng tr? thành m?t con ng??i quá khôn khéo trong ?ng x?. Hãy chính tr?c và có lòng t? tr?ng.

??ng ch? trích hay trêu ch?c s? l?a ch?n c?a ng??i khác. H? ??u có lý do khi l?a ch?n hay theo ?u?i m?t th? gì ?ó. T?t c? nh?ng l?a ch?n ??u ?áng ???c tôn tr?ng. Hãy s?n sàng ?? thay ??i t? duy b?t c? lúc nào trong quá trình b?n h?c h?i. M?t ngày có th? b?n thích Windows nh?ng r?i s? có ngày b?n thích Android. Không có gì là b?t bi?n c?.

9. LLVM

M?i ng??i say s?a nói v? Swift nh?ng qu? th?t th? tôi chú ý nhi?u h?n nh?ng ngày này ?ó chính là **LLVM.**



Tôi ngh? r?ng **LLVM** là d? án ph?n m?m quan tr?ng nh?t hi?n nay ???c ?o l??ng b?i nh?ng tác ??ng dài h?n c?a nó. Các Block trong Objective-C, Rust & Swift (hai lo?i ngôn ng? l?p trình r?t ???c yêu thích khi vi?t và biên d? ch theo kh?o sát các nhà phát tri?n c?a StackOverflow n?m 2016), **Dropbox Pyston, Clang Static Analyser, ARC, Google Souper, Emscripten, LLVMSharp, Microsoft LLILC, Rubymotion,cheerp, các ?ng d?ng watchOS, Android NDK, Metal,** t?t c? nh?ng th? này ???c h? tr? ho?c ???c xây d?ng t? LLVM. Có nhi?u trình biên d?ch ?ang s? d?ng LLVM nh? là ch??ng trình ph? cho khá nhi?u ngôn ng? l?p trình quan tr?ng nh?t hi?n nay. Cu?i cùng, .NET CLR c?ng s? t??ng h?p v?i LLVM và Mono c?ng ?ã s? d?ng nó. Facebook hi?n ?ang c? g? ng tích h?p LLVM v?i HHVM và WebKit g?n ?ây ?ã chuy?n ??i t? LLVM sang trình biên d?ch JavaScript B3 JIT m?i.

Th? nên, hãy h?c t?t c? v? LLVM. ?ây là "thiên hà" n?i mà s? c?i ti?n th?t s? ?ang x?y ra. ?ây c?ng là n?n t?ng cho 20 n?m t?i.

10. Hãy làm theo linh c?m c?a b?n

Tôi có c?m giác ch?c ch?n (dù không th? gi?i thích n?i) r?ng .NET s? tr? nên l?n h?n khi tôi xem s? ki?n ra m?t nó vào tháng 6 n?m 2000. Tôi c?ng có ?úng c?m giác ?ó v?i iPhone là nó s? càn quét th? tr??ng ?i?n tho?i di ?? ng khi tôi xem bu?i gi?i thi?u vào n?m 2007.

Qu? th?t, trong c? hai tình hu?ng ?ó, m?i ng??i ??u c??i tôi. Nh?ng c?ng hai l?n tôi tin vào linh c?m c?a mình thì chúng ??u ?úng.

Hãy nghe theo linh c?m c?a b?n vì bi?t ?âu may m?n s? ??n.

11. API là "ông hoàng"

API tuy?t v?i s? giúp t?o ra nh?ng ?ng d?ng tuy?t v?i. N?u API ch?ng ra gì thì ?ng d?ng c?a b?n c?ng ?áng v?t vào s?t rác, b?t k? giao di?n c?a nó có ??p ??n m?c nào ?i ch?ng n?a.

??ng phát minh ra thu?t toán b?o m?t c?a riêng b?n.

Hãy h?c m?t vài công ngh? server-side và ch?c ch?n Node là m?t trong s? ?ó.

Hãy ??t REST sang m?t bên và r?ng m? ?ón l?y **Socket.io, ZeroMQ, RabbitMQ, Erlang, XMPP** ; tìm hi?u tính n?ng th?i gian th?c (realtime) nh? là b??c ti?p theo trong quá trình phát tri?n ?ng d?ng. Th?i gian th?c không ch? dành cho các ?ng d?ng chat.

Và ??ng quên b?t ??u xây d?ng các bot v?i nh?ng API này.

12. Ch?ng l?i s? ph?c t?p

??n gi?n h?n s? t?t h?n. Always that. Hãy nh? nguyên t?c KISS (Keep it simple, stupid). Và ý c?a tôi không ch ? là ? c?p ?? giao di?n ng??i dùng mà là t?t c? m?i th? cho t?i khi b?n ??t ??n nh?ng l?p sâu nh?t trong code c?a b ?n.

Refactoring, unit test, code review, pull request, t?t c? nh?ng công c? này b?n có th? tùy ý s? d?ng ?? ch?c ch?n r ?ng mã mà b?n di chuy?n là ki?n trúc ho?t ??ng ???c ??n gi?n nh?t có th?. ?ây là cách mà b?n xây d?ng các h? th ?ng có tính ?àn h?i trong dài h?n.

Conclusion

?i?u quan tr?ng nh?t ? ?ây c?n ph?i nh? ?ó là tu?i tác c?a b?n không là v?n ??.

Mi?n là trái tim c?a b?n nói v?i v?i b?n r?ng hãy ti?p t?c coding và t?o ra nh?ng th? m?i thì b?n v?n còn tr? l?m.

Ch?c ch?n là b?n không bi?t ?i?u gì s? x?y ra trong vòng 19 n?m t?i nh?ng tôi có th? nói v?i b?n 3 ?i?u có kh? n? ng r?t cao s? x?y ra:

1. Ai ?ó s? ??t câu h?i trên Stack Overflow v? cách l?c ??a ch? email b?ng nh?ng bi?u th?c chính quy (regular expressions).
2. Ai ?ó s? ra m?t m?t framework JavaScript m?i.
3. Ai ?ó s? xây d?ng ???c m?t th? hay h?n c? LLVM.