

DIARY 2023

Abdur—Rahmaan
Janhangeer

*A collection of
LinkedIn opinion pieces*

Diary 2023

by Abdur-Rahmaan Janhangeer

build: 0.1.0

Slow != Not Useful

Pleasing People v/s Quality

Why Contribute To OpenSource

Programming History

Opensource is Tough

Types, Memory & Performance

OpenSource: Yet Another Way To Learn Rust

The World is Bigger Than FAANGs

Frameworks & Function

Formal 'Education'?

Side Projects

Independent Software Developer

*Am I The Only One Who Feels Like An Idiot
(Reddit)?*

Cringe Rustlang

Why a SQLite Book

ML is Maths

Ai & Fooling Humans

Why Startups Over FAANGs

Redis & Hyperloglog

Timelessness & Maintainers

Course review: NLP with probabilistic models

A Different Take On CVs

Business v/s Code-only Mindset

Software Engineering or Software Patterns

✨ Meetup question: How to choose what

OpenSource project to contribute to?

✨ Meetup question: How to land a hit job in

tech?

✨ Meetup question: How can we find time to dig into libraries and projects?

Who Is A Software Engineer?

Why In The FAANGs I Prefer Amazon

When No Coding Outside Work Impacts OpenSource

Modern Python Cookbook review

Back-of-the-envelope Calculations

Why do you love Python, sir?

Praise for Patterns of Distributed Computing

Database Performance At Scale, An Excellent Book

Having Intimacy With Employees: A Meaningless Criteria

What OpenSource Helped You Achieve?

Distributed Patterns Are Useless?!

Donald Knuth Clarifies

1. Slow != Not Useful

I've definitely heard a lot of rants about Python at dev meetups. If I were to listen to these people, Python seems to be the worst language on planet earth. It's also not a real language since it is a 'scripting' one. And yes, people must be very dumb to choose hashtag#Python as their main language. It also rAiSeS yOuR biLLs sInCe iTs SoO sLoWw.

It's interesting to note that one Python library is used by 75% of the top 25 Nasdaq-listed companies. If someone creates value fine. VC backing OpenSource seems to be the start of a new era. To watch out for sure.

2. Pleasing People v/s Quality

Just a thought that if you want to preserve quality, you tend to dissatisfy people.

Being the admin of the Machine Learning Moderated group (1M+ members, <https://lnkd.in/dfZtUgSd>). I constantly get DMs on why posts are not being approved.

If you are here to please people and accept everything, the group quality decreases and the audience vanishes.

Being focused also has the side benefit of attracting the right people.

3. Why Contribute To OpenSource

“People who work are too busy to do OpenSource”, you will hear most of developers say as to why they cannot contribute to OpenSource. Or, “OpenSource is for people who need to distinguish themselves and prove their worth”. Or, in the line of i have a Harvard degree, i don’t need OSS contributions.

We have forgotten why contribute in the first place. To make free software better (I like FOSS over just OSS). Or, to keep up with industry expectations and practices. Those two are formidable reasons. Building something meaningful is another.

If you don’t contribute to OpenSource, i’m sure you are missing a lot of fun as a developer. OSS contribs is growth on demand, at your own pace. It’s also an expression of creativity and more, as you maintain your artifact. A painter paints then he’s done. An OSS maintaniner does follow-up on his brain child.

4. Programming History

In programming knowing the history is wonderful in many ways. In other fields, the history might not exist to begin with.

History allows you to see through the evolution of something, understanding the components and motivation. It also opens up new possibilities. Some routes might have been left unexplored, some routes might be relevant now, compared to some 10 years ago.

Revisiting computing history is very refreshing.

5. Opensource is Tough

Contributing to OpenSource seriously is tough as you have to know what is happening.

You might rightly ask: But, don't you have to know the fundamentals to be in the industry? Yes, but, many techniques have been developed, and one project might take the techniques of other projects or some recent development.

To contribute to OpenSource, you need to have a grasp over the subject, that is, if you want to make sizable, meaningful and insightful contributions.

This is why i write articles, to serve as notes for me, and help future contributors.

6. Types, Memory & Performance

People associate types with memory and performance. For people who code languages this does not ring well. A typed language's execution depends on the program executing it.

For a dynamic language to include types, the main benefit is to try to help programmers out. It calls you out when you contradict your intentions.

This is why it's great to build conceptual models of things. This tallies with a natural method of teaching. You explain things grossly. Then for each implementation, give the specific details. You cannot make general statements about how programming construct x operates beneath the hood.

It also boils down to the fact that people who code things up iterate over an idea then add their own spin to it.

7. OpenSource: Yet Another Way To Learn Rust

I read the [Rust book](#) twice. Also [Comprehensive Rust](#) from Google. Things did not click as i forgot ownership fast when digesting other topics. The Rust book has a lot of concepts to cover. I lost a lot of time starting again. I really wanted to get my feet wet and started programming on some playgrounds as well as trying some [Maturin](#) examples.

Playgrounds are fine. Until you try to code something on your machine. I thought rustc was how you built things / run scripts. I tried cargo finally and everything worked like a charm. At this point i decided to code something fun which would have the right balance between motivation and fun. It should also ideally have some stuffs i can do, so that i can focus on upskilling in Rust rather than figuring out what to try out next.

I decided to port Python's library called [rich](#) over in a crate called [richterm](#). Since it's a port, i had a syllabus to cover, a lot of todos to tackle already. This in itself carved a path where i learnt Rust progressively. It was like learning Rust by tackling challenges meaningfully.

In the beginning i used only functions. I wanted the [print function](#) to support both arrays and vectors.

This introduced me to rust generics. I also got a gentle introduction to `&str` and `String`. Then i saw the need for classes, i explored `struct`. To hold data, i explored `hashmap`. Then attempting threads for `spinners`. I even have the luxury to pause and continue working when i like, for example while coding the panel feature by saving it as a `pull request`. I also got people `helping out`.

Ah, i also got used to defining modules and adding a crate and use it. I discovered `crates.io` and how to add projects to use.

I could not have asked for a better way to learn Rust. The community is willing to teach. Building an OpenSource product introduces you to some good practices. I'd say think of a nice-to-have thing and give it a try. I even learnt about `cargo fmt` while `contributing to libSQL`.

I'd say this way is far more lively, real and entertaining! I am not a Rust expert but it motivates me to keep pursuing my Rust journey.

– Reddit

8. The World is Bigger Than FAANGs

I have the impression that what distinguishes FAANG companies from others is primarily because of it's popularity, and it's hiring rounds.

I don't see all employees of FAANG companies work on exciting stuffs. It's a very tiny proportion. The rest are maintaining screws and bolts. It's like 50000s people are maintaining a car. Yay i'm a great someone because i maintain a micro screw!

I think it's quite boring in terms of personal growth. Yes you have great insights, you see how 5 of the world's largest entities do stuffs, you touched 1000s of lines in x time that put app y in the hands of million of people, but you are missing on cutting edge tech. It's impossible for each and every 50000 person to be involved in cool tech as the whole company will go bust overnight. The bigger the mammoth, the greater is the need for stability. Most employees interact with the 'stable' part. You sure can enjoy where you want.

There are big companies with tougher rounds than FAANG but not with the same level of impact. Same with pay. Same with perks.

Everytime i see a vid on Ex-FAANG this or that, i'm like hum hum what the person was doing exactly. Oh you helped one of the FAANG do this or that or you are at that level good but unless you are in the tiny percent of people who drive innovation forward

(FAANG have their own problems which a tiny portion of people are chosen to work on, but again problems at scale is not a FAANG exclusivity), i don't see the merit of being ex-FAANG.

In sum i don't believe that great people come only from 5 companies. The world of software is diverse. Stunning pieces drop from many big and small companies around the world. If you open your horizon you will see that the world of software is bigger than and not restricted to those 5.

9. Frameworks & Function

Frameworks are the enemies of knowledge, but, they do accomplish the purpose of knowledge. Frameworks require knowledge of their own, they create a virtual space ruled by them. People, most of the time, are unaware of the underlying space. Problems and solutions are communicated in reference to, and expressed in terms of the framework. A web framework for example. People don't care about packets, headers and protocols. Any progress in the protocol realm is ingested in the frameworks. If one framework fails to adapt, another framework appears. People are then sure never to leave the land of frameworks.

Due to the growing complexities of the software world, frameworks hold a certain legitimacy. If you need to deliver something, frameworks help you reduce delivery time.

This said, since frameworks are required by jobs, most of the time developers don't care about what's going on beneath the hood. This over time erodes skills and creates a world where people no longer know how the world works. Bright people are those who dare understand and spin another take at problems or derive optimizations.

Curiosity and experiments are two major motivations to keep learning. We should actively learn even if we don't see the need for it, because this creates a pool

of ideas and broaden insights which can be used to tackle problems and artfully carve programs.

This is why contributing to OpenSource provides a constant stream of serious, meaningful, and breathtaking side-learning.

10. Formal 'Education'?

I am for formal education, but, not if it can be replaced by a radio sitting on the table in the place of the lecturer, which could have also been replaced by a youtube video.

The software world moves at reckless speed. Institutions attempt to teach the basics. But, the basics themselves evolve tremendously. Fortunately education is scripted to fulfill industry needs. There is not much need to be proficient if you are going to work for those companies for which the courses have been designed for.

They are boring of course. The most funny thing of all is that to make it at top companies, the basic level of cs education is not enough. Right off the bat, people are unfit for entering these companies. They need some 2, 3 months of job preparation to crack algo rounds. But wait, are you not supposed to be learning algos in cs?

I think there is a great disparity between teaching and the industry. Some institutions attempt to fill this gap by sending students to companies. Sometimes students are left on their own to find their own companies for internships. Are they taught how to look for internships or should they be looking for internships? I mean what are they paying for? I pay to learn at x institution which sends me to learn at z company. z company is considered a nobody, why is x institution not able to impart the knowledge they are being paid to impart?

There is a great deal of learning to be done individually. I won't be surprised actually if learning at an institution means learning on your own 90% of the time. I'd call it thieving if i did the effort which i am paying to help me with and, i don't get the credit for. Some institution put their name and take the credit of my work.

The learning materials also are not very interesting. Many great people have clear minds and clear explanations. Not the case with most teaching materials. We don't need fancy illustrations, but, we do need neat, concise and clear explanations. The materials are of abominable quality, and people who have been sufficiently exposed to abominable quality materials certify that these materials are indeed excellent.

The mass production of industry consumables is bound to quell a lot of curiosity and passion for a topic. There is the rule of demand and supply but, if someone studies a topic in a horrendous manner through which they are disgusted to look deeper beyond job requirements, i see it as an educational failure.

11. Side Projects

I don't experiment and explore because i have a job. A job which confines me to the same legacy codebase and tech stack for the last 5 years. And when the beloved company sacks me, i find myself in a different world.

I spend 8 hours working. I don't invest in my upskilling. The uncaring attitude of the company forces me to not care about myself. And i promote this lifestyle to others too.

Work without experimentation is weird. Maybe in some fields workers should not keep themselves updated or they don't have the sufficient background to experiment but, software is such a field where it makes sense to know what's going on and explore.

And, having great employees benefit the company too. Unless the company is big enough to afford to tell people to code and stop talking, this is a missed opportunity. Some big companies love their employees not knowing outside tech, so that they might feel that their worth lies in remaining at the company.

This is why contributing and maintaining OpenSource is a great way of experimenting with ideas without fear.

12. Independent Software Developer

An independent software developer is another word for a freelancer who is more geared towards contracts. It usually involves projects for a longer period of time.

This might seem contradictory to have the word “independent” and yet work with companies. If you see it like this then it’s weird for someone to work independently i.e. with no clients whatsoever XD.

13. Am I The Only One Who Feels Like An Idiot (Reddit)?

ruslang is one of the hardest languages around as it simply has lots of concepts to cover. I learnt C++ as my first typed language and i feel that rust complicates the plot with lots of it's own additions. C++ feels like the Python of typed langs.

Instead of manually managing memory, many languages rely on a garbage collector. Rust allows you to do automatic memory management by inventing a story about ownership. It also occasionally contrives some traits and lifetime issues for you. Even seasoned developers are poked around.

Meanwhile rustfluencers have been pumping the language as if it's the best hit since bread and butter. The reality is that it's a language with a lemon taste. People who tasted it attempt to lure others to the same trap and pretend that you must be mad for having a continuous unsavory aftertaste. Users also fear of being called dumb for having difficulties with Rust.

Back-end wise it's brilliantly designed. But the usage experience is quite horrible given the precedence in language design we have now. Syntax-wise it adopted nice features, even promotes nice conventions, but, the overall end-user experience is


*just bad due to the concepts it invented. Rust just
needs to write a better story.*

14. Cringe Rustlang

Rustlang is an out of space language, the future of programming. Obviously for such a language, you must unlearn the bad from your previous programming life.

Devotees of the cargo community pass through an initiation ritual through which they are taught mind-shattering wisdom. They learn the pure doctrine that humanity has been doing types wrong since the beginning of time.

They learn of the glorious string type that has been so brilliantly designed that it's actually two types which is actually one type existing in two embodiments but which remains one as both preserve the essence of the type.



*LMAO at all the rust developers calling
to_string on strings WTF cringe - HTMX*

15. Why a SQLite Book

One strong reason to write is for self improvement first. Then to benefit others. When writing articles, i know very few people will read them, but, i am not too keen to go with the hype.

I like SQLite a lot. libSQL is a remarkable fork. Though i wrote my book on SQLite internals, libSQL caught me offguard on some aspects like the WAL format details. It's a core piece of their syncing strategy. Writing this article was like re-writing a book once more. It was a combination of several maintainers knowledge.

But for people who are in the field, it rings well. If you go with the hype, you will find yourself with hyppy people. If you are interested in people working on databases and languages, you will find yourself with those people.

article in question: <https://lnkd.in/enVGcyc>

16. ML is Maths

The more i learn about machine learning, the more i see as being a mechanical process. There are two school of thoughts in the computing world, one that says that computing is a field of it's own and one that says that it's an expression of maths.

Today, the gold rush of the industry is Ai. Ai is a vast term but, when companies say Ai today, they mean machine/deep learning. The core of ML/DL is maths. They essentially have a formula which they tune according to the training data to output a result then use it to compare the result of new samples. Or, they learn patterns and output variations of it.

It is driven by numbers, using a variety of techniques. And probability, a lot of it. The goal of computing has been automation. Automating calculations is the core of it. The recent variation of Ai has been a sublime illustration that computing is indeed maths.

This might seem weird but taking advanced ml/dl courses at the end of the day boils down to learning and understanding formulas by deriving them from scratch. It also includes knowing how to use a tool/library to do so. Also about techniques to patch results we don't want. It's very boring, it's dry maths at work.

17. Ai & Fooling Humans

Ai is about fooling humans. To decide whether an Ai behavior is good, we compare it to an expected behavior. Let's say we build an Ai that simulates a philosopher. If it interacts correctly, and gives great answers, people will deem it as good. More importantly, if it mimics human experts then industry voices which is experts-backed will deem it as good.

Let's say we want to test whether or not Ais are sentient. What do we do? We'll first define what it means to be sentient. We'll find those behaviors which sentient beings express and compare it to the Ai. But, the problem is that developers will go through the same process i.e. finding according to what criteria people judge sentient beings and code the Ai to emulate those behaviors. Whatever ways of interaction you put as criteria, the Ai will try to emulate that. It's exactly like an enhanced mirror.

But, we are not foolish enough to decide that a mirror is indeed a person because we can see a human over there with human-like interactions. Ai means Artificial Intelligence. Meaning it is a field to monkey human intelligence.

18. Why Startups Over FAANGs

That's one take where working at startups might be more interesting than staying at a FAANG company. The company might be surfing on billions of USD, it might even have the same service internally but, it's also about the focus, passion and purpose.

Scott worked at Google. Google had Cinder internally. It was a cloud IDE that the mega giant built. It had loads of infra to test on. Bright minds to work on it. But, Replit was more appealing. The problem is hard to tackle but there not much interest from Google to do it, it's not that they can't, it's that they are not interested.

Solving problems in a niche field has an irresistible appeal. Specially if you are at the very bleeding edge. Specially if the startup has been consistently delivering and showing proofs of foresight.

 <https://lnkd.in/eZFBU-C4>

19. Redis & Hyperloglog

Redis is amazing. It seems very ordinary. It describes itself as an “in-memory data store”. So far so good. But, it’s also used as “database, cache, streaming engine, and message broker”. Keeping this list in mind gives you an idea of what to expect when trying to understand the what and how of it. It’s like having a lorry, car, bus, plane and rocket all in one vehicle.

One data structure it uses is HyperLogLog. This was not invented by Redis but it uses it. LogLog is beautifully summarized by the authors:

“Using an auxiliary memory smaller than the size of this abstract, the LogLog algorithm makes it possible to estimate in a single pass and within a few percents the number of different words in the whole of shakespeare’s works”

This means it provides a method to count the number of distinct elements (cardinality) a set, here a file, contains. LogLog gets its name from the memory needed, LogLog Nmax where N stands for an upper bound on cardinalities. It was also designed to be used distributed or parallelized XD.

It’s used when storing the actual unique data is not possible due to the enormous amount of it. It estimates how much unique data there is. It initializes a table called register of binary items, hashes elements, inspects the binary hash of it in

terms of leading zeros to determine where the register should be updated. It then uses a formula to calculate the number of unique items; based on the longest number of leading zeros it observed and the elements of the register. It reminds us of how a bloom filter is implemented. The size of the table/register it used for the shakespeare's works is 256 bytes of 4 bits.

Super loglog is the best version of the loglog algorithm. Hyper loglog is a more accurate version of it. There is a lot of stats at work in this, for once, an interesting algorithm to dive in.

So, Redis allows you to use Hyperloglog as a supported structure. If you need to count unique elements which occur in massively great numbers, it seems a great option!

20. Timelessness & Maintainers

Maintainers aim for timelessness. They aim to provide a library that maintains relevance across an ever-changing landscape. The base tech changes. Operating systems come and fade away. Languages come and go. Timelessness is difficult to achieve, unless the essence of the library is captured and ported or re-wired time and again to ensure a living embodiment of it.

But, a more pressing and easier to solve challenge is aiming for relevance in the original setting the library was designed for. The first enemy to avoid is the library itself. If version 1 is a contender to version 2, it breaks the user experience and puts the library authors on a bad reputation watch-list. Backward-compatibility involves lots of efforts. it also requires much foresight to avoid a break in architecture for example. Forward compatibility is also an interesting option.

Sometimes library cannot achieve a top-notch relevance as the environment they operated on changed. Let's say library x was written for an operating system which broke a consistency aspect. Library x cannot run on all releases of the operating system not due to the library's fault, but due to an external factor. Still, it strays from an ideal the maintainers were aiming for.

Htmx, a normal, node-less OpenSource JavaScript library was recently included in Github's first OS

*funding programme. Alexander Petros, a contributor,
wrote this brilliant piece on timelessness.*

21. Course review:

NLP with probabilistic models

Stats is always great! I liked this course, no wonder it has a 4.7/5 rating, the highest of all the 4 in the specialization.

week 1: Essentially Norvig's explanation on how Google built it's spell corrector. Norvig's explanation is better in my opinion. This is one great illustration that problem solving beats knowledge hoarding where Norvig was amazed his maths peers had no hint on how it worked.

week2: Markov models, emission matrices and lots of viterbi. Know and code.

week3: Ngrams and dealing with unwanted cases. Know and code.


week4: Very exciting. The part about creating embeddings from scratch is 🔥 . However, they did a very very very poor job at explaining classical concepts like gradient descent. it's not because that they did not code it from scratch, it's because if they are really explaining it for the first time, it's bad at making people understand what it is. Relu and softmax were top notch though. I think i followed most vids with attention, very engaging!

22. A Different Take On CVs

I have a different take on CVs. If anyone asks me for a CV for recruitment, i send them a nice-to-read but up-to-the point document, which focuses on opensource projects. Respected recruiters and esteemed companies in Mauritius of course bin it straight away.

It's a way to avoid working with people who cannot analyse documents or who set up barriers which prioritize bureaucracy over business needs.

I like to work with people who are simple and close to their objectives. If someone is serious, everything flows smoothly. I got 1000s of usd in few lines of messages, even as short as 3.

 Do you do x

 Yes

 I propose y

As bland as this.

My CV brought me exactly Rs0. My Github on the other side, brought me Rs100k+.

There are sharp people out there who have different ways of meeting their business objectives. Sure, if your aim is to get money via a CV, you'll optimize for that. At the end of the day, you must decide whether you want to get money or get money via a CV.

CVs are like a game of SEO. If your content is good, you'll give up playing with the algorithm. The algorithm will find a way to include you.

23. Business v/s Code-only Mindset

Having a business mindset is better than having a code/engineering only mindset.

With a business mindset you care about the objective. Having a code only mindset, you are focused about quality, metrics and practices. A business mindset makes you care about end users. A developer mindset has the tendency to focus on technically glorious features. A business mindset shifts feature direction to one that aligns with the objective. A business mindset cares about people at the expense of technical flashiness.

One easy way to develop a business mindset is OpenSource. Launching an OS project is like launching a business. You need to find a worthwhile problem to solve if you want people to use your product. Usage count is like your revenue. Once launched, you need to provide the best usage experience. You need a lot of marketing. Your product might be good, but if people don't know about it, they won't use it. You need to invest in documentation and fixing bugs which corresponds to the customer support experience. You also need to eye competitors and decide if competing features are worth integrating. We also need to see where we can push innovation and of what impact. A breakthrough requires a lot of research and effort, will it be relevant to users? One interesting addition is that, if

you want industries (big players) to adopt your OS product, you need to provide good quality ones.

That's one huge plus when contributing to OpenSource as the repo owner. You cannot take full control of a project when you just contribute. When you own the repo, you own the responsibility and reflect on mistakes. You also learn how to deal with people (contributors) as well as the board (co-maintainers). You need to also infuse in them the business mindset as they will tend to have a code mindset ^^.

24. Software Engineering or Software Patterns

It is very curious and concerning that a lot of books on software engineering focus on ... software patterns.

Engineering a robust piece of software requires a lot of stuffs. One of the main characteristic is being bug-free and providing working features. This requires the addition of valid and vetted bite-sized pieces of code. There is a need to address the approach to development.

There is also the issue of foresight. This is an indispensable aspect. Foresight often shape the direction of the project and prepares it to deal with the expected unexpected. Field knowledge contributes a lot to foresight.

Then we need to develop clean apis. There are very few books dealing with how to develop clean software apis. Great library usage experience is not that which is logically correct, which can be composed. A greatly architected library might have a poor api usage experience. It is about how easy customers find it to use it while providing powerful features. I asked a community of brilliant people to recommend resources, there are few resources as most books focus on clean codebases.

Then there is the aspect of security. People learn about security as an after-thought, imbibing post-mortems. This should maybe be a fundamental piece of education. Security and web frameworks should go hand in hand. There are a lot to security, devops sec, ci/cd sec etc also forms part of it. True that frameworks limit a lot of issues but, you should be intelligent enough not to rely on frameworks or be insightful enough to avoid pitfalls or add additional layers. Or even know about issues which the framework cannot address.

There is also a boring part which requires writing code to thrive in hostile environments. Malware authors teach software engineers lessons after lessons. People often tell this and that cannot be done. Malware authors often teach brilliant lessons on distribution and stability. Even i am amazed at how they write Python-based viruses and RATs which run undetected and the ease with which they distribute those to different OSes. The software world is still like well Python this and Python that. Malware authors take the patience to write code to deal with obstacles in their way.

Software engineers are often picky about code details at the expense of shipping a good product.

25. ✨ Meetup

question: How to choose what OpenSource project to contribute to?

First of all, there are different approaches to OpenSource. You have the hobby approach and the focus approach. Here we are talking about the focus approach.

Have in mind that you'll invest 6 months on a project. Find out what drives your motivation in computers. Is it cryptography? Is it databases? Is it data? Is it machine learning? Is it operating systems? Pick one field. Next find a popular OpenSource project in that field. If it's machine learning i'd pick something like PyTorch. Next verify if the project accepts external contributions.

If you want a dead sure way to land meaningful contributions, you need to invest in a project fully. There are two options: either you have a clear indication of what you want to contribute or, the sure way, find something meaningful to contribute. Here we are talking about an approach that will also find us an area to contribute to.

This approach is the hard way, involves a lot of effort but, it's worry free. The first step is to understand the theory behind the project. How the project works internally. Find resources about

internals. They'll discuss important aspects of the project. The codebase will be centered around those ideas.

The second step is to have a codebase walkthrough. Start exploring important parts of the codebase. Read on until you are familiar with the overall structure. Read as much code as you can. By now, one super interesting opportunity is code comments. If there are not enough meaningful comments, add what you learnt. Also by now make sure you know the programming language the project is using a bit. You should clone the project and get it running. Feel free to make some changes and run it. Messages are one awesome way to tinker around.

Document your learnings along the way. If you followed the above recipe, you should be able to contribute to any project you like meaningfully. I wanted to contribute to SQLite, but it does not accept contributions. On LinkedIn saw a fork. I decided to contribute to it. I knew 0% of SQLite internals. I learnt about it. The fork added gazillion of features. I took the pain to learn them. The libSQL/Turso team helped me as much as i wanted to. I contributed docs about the internals. I also landed a PR about comments. I learnt rustlang to contribute to it.

The journey continues ...

26. ✨ Meetup

question: How to land a hit job in tech?

How to find awesome companies to work with? The short answer is: don't find them, let them find you! We live in a connected world. We have global competition. It is not necessary, but, it is an interesting option to have skills that are valued at global level.

One way to pick up skills is to go to university. You do work on awesome projects, you have great peers and connections, but, often time people accept you only on the basis of the university's reputation. You are also evaluated and socially graded according to it. There is a hiring bias of whether or not you walked on the sacred grounds of Stanford. There is also the business of recognition. Not all universities are recognized everywhere.

The experience of workplaces differ. Levels are not the same everywhere. They do not mean the same thing. A senior someone at company x might be a junior someone at company y. Time in a field is also bad metric of evaluation. 10 years or 3 years don't matter as much as the skills do. What projects you worked on can be gauged via interviews.

One way to evaluate yourself according to global standards and set yourself at such level is through OpenSource. Due to greed, companies value

OpenSource a lot as it let them have some free, stellar-quality assets to start with. If you started furniture production, you have some initial costs associated which is related to the base material itself. This is not so if you use OpenSource.

The software world nowadays turn around OpenSource. The biggest hit products of our time are open. Almost all servers on the internet use OS. Almost all websites use open techs. Even if an ecosystem is a walled garden, they taste the sweet fruits of open components. Currently, you cannot beat the quality of software that a bunch of mostly unpaid people work on in their spare time.

The thing is, if you become an integral part of this ecosystem, if you become a core piece of it, you become a valuable resource for companies operating within that sphere. And with remote, this opportunity is available wherever you are. To start and contribute meaningfully, you need to help yourself get started. It's a continuous journey; albeit, a delightful one.

27. ✨ Meetup

question: How can we find time to dig into libraries and projects?

This is a difficult and easy question at the same time. I feel that if someone considers something as important, in 99% of the case, they will devote the time for it. There is a difference between believing something is important and knowing that something is important. If you know and accept but don't believe, it won't lead you to action. If you understand and see that exploration can be good for you, you will do it.

One way to optimize contributions is to find something related to what you do at work. If at work you work in the data field and use pandas for example, it might be a good opportunity to look and contribute to pandas itself or polars. This is a nice approach as when contributing, there is the initial field learning phase and knowledge of the tool. If you choose something in your field of work, you will save the time to learn a new field.

This also has the double advantage of enhancing your working skills. When contributing you go to a level deeper than usage knowledge. You dive into the rationale behind decisions, problems facing the field and how they are tackled. It is also the time to get

a bird-eye view of the field itself as you will be looking at players in the field to contribute to, and while contributing you'd sometime see how alternative projects tackle problems.

If you cannot find time due to work demands, then maybe look into alternative jobs or, in upcoming contracts try to find a way to include this time or, find a flexible schedule that allows you to explore. Experimentation is important for personal growth.

28. Who Is A Software Engineer?

The software world is a mind-boggling riddle for some people who are education-focused, to define who a software engineer is.

They think that for people to work in this field, they must have the required accredited license from a reputable institution else, they consider the person to be a pure 0. They paid a lot of money and sweated a lot to get the degree / license and it's understandable that they will feel very uneasy when they find out who a software engineer is. It's even more understandable if they are still footing the loan bills. It's a lot of fun to engage with such people as they tend to bog down others with their academic credentials.

Someone took a computer science course. Another one took a software engineering course. Then, according to them, the first person is a computer scientist and not a software engineer. The person is a complete noob in coding softwares just because he does not have a software engineering credential.

Someone finished a bootcamp or glovecamp. They managed to crack Google's entry-level rounds. Google employs the person as a software engineer. But according to these people he is still not a software engineer as he did not take a software engineering course.

Now the person works for Google. He climbs the ladder. He even has time at some cool department in there. He shifts to another company or startup. He also published ground-breaking papers. He is still not a software engineer as he did not take the divine software engineering course.

Someone is an OpenSource contributor with no degree whatsoever, over time he becomes the maintainer of a Linux component. Over time he becomes a core maintainer of the project. He is also a maintainer of several other prominent projects. When such education-minded folks need Linux expertise, they consider this person as a 0 as he did not take a software engineering course.

Someone authors a language. He has a doctorate in compiler theory. When their company needs expertise to work on their codebase written in this language, they won't consider the author's profile as he does not have a software engineering degree. No way they'd employ him as a software engineer at their company.

At their company, they are also wary of developer toolings. They definitely verify whether the web framework that they use, the terminal that they use the operating system that they use, the browser that they use, the email client that they use has been coded by people who have a software engineering degree as they must be sure to run their business on pieces oozing from competency. They investigate whether Google, Microsoft etc indeed employed people with software engineering degrees to build those components.

Those respected software engineers are indeed brilliant. They meritoriously earned their

nitwittedness degrees. They should work on getting their next license: The intelligence degree.

They also don't use LinkedIn as they are unsure whether people who coded the Ui and the backend have software engineering licenses. So, they won't see this post. What a relief!

29. Why In The FAANGs I Prefer Amazon

From among the FAANG companies I like Amazon the most, just because I have always met Amazon folks in cool places I spend time.

I've had amazing conversations with Amazon people on Twitter. Even had a podcast with an engineering manager. One of the core contributors on one of my OpenSource projects is from Amazon. I took one amazing class and some of my classmates were from Amazon. I had great conversations with them on LinkedIn.

Amazon is often depicted as being somewhat toxic. The environment is harsh in terms of performance reviews. The people are industrious. People over there seem to be working twice as hard as at Google. They get bored when coming over.

But, they are very social people, like to make friends and jump on cool initiatives and help out. I like them a lot!

30. When No Coding Outside Work Impacts OpenSource

Foreign companies are welcomed in a country except when they have the strict policy of crushing the local development ecosystem. More precisely, when they actively suppress up-skilling.

Coding outside of work is frowned upon as this might lead employees to work with direct competitors. Let's say this is a legitimate reason, then, there is no reason to ban OpenSource contributions. This level of ownership of an employee's resources is plainly ridiculous. The employees cannot contribute to projects to help cool initiatives nor can they invest on themselves to become better.

Contributing to OpenSource strengthens a country's developer ecosystem for free. Imagine if all companies had to forcefully upgrade a country's ecosystem, they would whine about it. Local companies benefit when they have great resources around. When foreign companies set foot in a country with cheaper labor, they enter with a workers-squeezing mentality. They want only acceptable output at cheap costs. Even if the developers are burnt in the process. It's an accept, suffer, or quit policy.

If the said entities employ thousands of workers, their policy becomes the norm in the industry. Oh you do OpenSource? We won't hire you precisely because of this. If employees did not know or cannot contribute because of time, fine. Else, if this surprising policy exists as a norm, good elements won't contribute to a country's corporate culture. The ecosystem is then not earmarked as an investment destination.

This whole thing ultimately affects developer happiness. This policy is a red flag. If you are cornered in terms of money, you will accept to hold red flags. But this will make you suffer in the long run as an individual.

31. Modern Python Cookbook review

Packt sent me a reviewing copy of Modern Python Cookbook by Steven Lott. This is a wonderful training material which can be used to get a solid foundation in Python after knowing the basics. I'm pretty impressed.

The book adopts a nice approach. It raises questions and solves them with generous explanations. It includes commonly-encountered scenarios and makes use of the standard library. It also includes packages when necessary like Flask and pytest. It's the perfect book for teams who want to use Python immediately while getting well-grounded information.

Since this is Python3.8, the typing information is not to be taken too seriously and some newer features like structural pattern matching are not included.

It's more than a cookbook. It cements concepts and provides industry-level guidance. Packt quality has really improved, both content-wise and format-wise. Code snippets are highlighted, which is amazing.

It's a delight to follow along 👍

32. Back-of-the-envelope Calculations

Often you might come across these funny numbers which some authors assure it's pretty important to know about. They don't make sense until you need to do back-of-the-envelope calculations.

How do you decide whether proposal A or proposal B is faster? You will think in terms of time. One system uses a remote Postgres as a key-value data store served behind a web service via a monolith and the second uses a local Redis served via a micro service architecture. To answer questions like these, first, it is imperative to know how systems work. It is sufficient to know the big components abstracted. If you are a fan of reading OpenSource codebases, this should be even easier. A monolith for example is faster than microservices just by examining network trips. In-memory access is faster than a remote call + writing to memory/disk as Postgres has a server on top of the engine.

You can gauge an answer based on network trips, memory, and disk access. Now let's say you want to have concrete values to base your estimates on? That's where these funny numbers come in. They pin down a number on the operations. Reading an image for example is equal to disk seek + time to read the image. Guessing the time taken to transfer it is just the filesize / transfer rate, and so on.

The first person I saw talking about back-of-the-envelope calculations was Jeff Dean. Great chap!

33. Why do you love Python, sir?

Interesting question, indeed. I like Python because it's for once, a language designed for humans to use. A language should be designed for humans before the machine, as the primary users are humans. Python is optimized for reading. Its conventions are designed to be as close as possible to natural language, in this case, English. It identifies blocks with indentations, without the need for braces, just like techniques for differentiating pieces of text. It also tries to clear out concepts that could result in the language looking like hieroglyphics.

Another lesser-known reason for me is that it has an awesome, compelling, ground-breaking spirit in terms of language design. The community has nice gut feelings about innovating cool constructs. The for in loop for example. In the quest for better end-user experience languages often copy Python features. Being involved with these awesome folks and tracking proposal discussions gives one a well-grounded view of the spirit of programming. The Python team I feel is years ahead in terms of molding concepts into fine APIs, built on years of excellent legacy in this field.

Cool language, cool people. When a language is accessible to humans, cares for humans, and listens to humans, we can expect the halo around the language to be human-friendly. The Python community is an excellent, welcoming, and attentive

community. Collaboration and help for events are extraordinarily smooth.

Since a language cares for humans, humans use it, a lot. Much to the consternation of wildlife hunters and killers, Python remains alive against all odds. People try to shoo it away, try to throw it outside of the window, but it remains around, firmly put. Since humans use it, they produced lots of artifacts. The Python ecosystem is vast and large. Many packages in certain fields exist only in Python. Packages are conveniently named, designed, and improved. This also indirectly speeds up the development cycle.

Even if tomorrow I switch to a typed language, I'd still keep in touch with Python to remind myself about what good language design means and keep myself inspired. It's about keeping the body of API intuitiveness in my mind fit, in a world where newer mainstream languages adopt a disgraceful and deplorable approach to the compiler front-end experience. It's as if the world is run by ice-age minds, who, in feeling closer to the machine treat themselves and most importantly others to a machine-minded experience. If pain is a guiding principle in determining correctness, we'd have fun talking about how some languages propose a case for adoption while inflicting upon people the wincing pains programming had to offer when it was still fumbling along the corridors of the dark ages of pragmatism, based on a language invented by the admission of its own authors as a joke.

34. Praise for Patterns of Distributed Computing

What makes this resource excellent is that it presents up-to-date distributed patterns inspired by opensource. It's not some dusty, old, theoretical, academia-favourited accounts of concepts. It references actual implementations and newer techniques found in the wild, with the accompanying context i.e. the projects that implement it with the accompanying nuances.

It's intensive, refreshing, and very insightful for system design. I don't expect system design surprises by now, but I like to build things and understand how they work. From that perspective, this book is an awesome resource.

Thanks, Unmesh Joshi for taking the pain to collect those gems. And thanks Demitri Swan for sharing. Demitri consistently takes the time to post interesting computing items and offers personal mentorship in batches from time to time. Worth a follow IMO.

Free to read online: <https://lnkd.in/enrrAibF>

Oreilly (10 days free): <https://lnkd.in/ePfhTbqq>

35. Database

Performance At Scale, An Excellent Book

Read Database Performance At Scale, a free book by ScyllaDB folks. It offers great insights into ... performance. At some point, you yearn for performance boosts, this is where this book is helpful.

I'm glad I covered distributed systems before this one, as sharding and data made more sense. Knowing libSQL and SQLite internals also helped a lot like knowing the particulars of log-based replication. Drivers and data structures and algorithms behind DB internals were a great read for me. It's also interesting to see the techniques mentioned here being implemented in libSQL like getting the data closer and user-defined functions. I knew operating system internals but not specifically Linux, had to google a few concepts. Nice point about containerization is not for free as os-specific optimizations might get wasted. It also covers monitoring and benchmarking.

Glad it has a practical touch to it, helps people dealing with dbs much as it has ready-to-implement items. A delight for system design folks as well as any curious individual.

Thanks, Piotr Sarna, Felipe Cardeneti Mendes, Pavel Emelyanov, and Cynthia Dunlop for producing such a

treat for database people. I can imagine the work that went into this one. Even me, with a great share of the book's info in my pocket, I learned a whole lot.

Download: <https://lnkd.in/eUZsgGKi>

36. Having Intimacy With Employees: A Meaningless Criteria

If employees state that they have intimacy with other employees, then you are suddenly in a great place to work?

“camaraderie takes up a huge aspect of our survey which is required in order for you to become a great place to work” [1]

One of the pillars of Great Place To Work® Mauritius is camaraderie.

“Friendliness and a sense of community”

“We’re all in this together” [5]

I guess this is my company, my family (until they lay you off) type of thing.

Employers would adore that label for sure. So, your colleagues are really your intimate[2] family members.

“I think it’s so important to have personal relationships at work and doing things even outside of work with your colleagues” [1]

So, in order to work at a great place to work, you have to accommodate in your life, a few new brothers, sisters, husbands, wives, people whom you share intimate relationships with.

“Intimacy is different from having “intimate moments,””[3] Zeynep Tozum tries to quote in “Intimacy in the Workplace, a radical idea” (1994) but people do watch for “... Prevention of Sexual Harassment at Workplace, Grievance Redressal Policy, Equal Opportunity Policy, Business Ethics, and Morals Policy. These policies ...””[4].

So, what it means to be intimate without being intimate? It means: “One should maintain more distance with colleagues than what they would maintain typically while having a conversation with their family members.” [4].

So is it about being close or not being close? Use a word then try to back out. Intimacy is a bundle of contradictions. Even Great Place To Work has shockingly few resources on how to effectively build camaraderie. Just search for it. I found only water cooler, skit and club. But, what is camaraderie exactly? “Intimacy Is Hard to Define” says Zeynep.

So, are you being certified on a pillar that is hard to define?

GPTW defines intimacy as “the extent to which people can be themselves and count on each other [9]”. To accomplish it they do things like (Reddit) “having inclusive teams and programs means that individual values and different perspectives are embraced and sought out. These viewpoints help employees challenge themselves and one other to think unconventionally”[9]. But, what a blunder! if you are not in a (love?) relationship, self-disclosure may not be essential [10]. Besides the point that it took a mistaken definition of intimacy from Reis and Shaver.

Camaraderie is composed of intimacy, hospitality and community [7].

*So, to be certified, you do have to belong to the company [8] (commun*ism?), and share intimacy with other employees.*

I could not find a clearly labeled article on creating intimacy from GPTW, save the Reddit one.

I definitely would like to see more on intimacy from GPTW. If you choose to promote a criteria in a country, you must have a clear idea of what you mean. Also GPTW, do people want intimacy?

👉 [1]

<https://greatplacetowork.me/webinars/camaraderie-and-its-impact-on-productivity/>

👉

[2] Ally is a large company with an intimate feel <https://www.greatplacetowork.com/best-workplaces/companies-that-care/2023>

👉

[3] <https://markkoenigsberg.medium.com/intimacy-in-the-workplace-a-radical-idea-9b9bef257edc>

👉

[4] <https://hr.economictimes.indiatimes.com/news/workplace-4-0/workplace-intimacy-a-boon-or-a-bane/88183184>

👉

[5] <https://www.greatplacetowork.com/resources/blog/purpose-at-work-is-only-profitable-if-you-do-this-one-thing-study>

👉

[6] <https://www.greatplacetowork.com/resources/blog/better-together-how-we-at-great-place-to-work-are-growing-closer-at-a-distance>

👉 [7]

[https://www.greatplacetowork.be/images/2023/Website/Great_Place.](https://www.greatplacetowork.be/images/2023/Website/Great_Place)

👉 [8]

<https://www.greatplacetowork.com/resources/blog/best-workplaces-in-europe-2023-flexibility-well-being>

This culture of belonging

👉 [9]

<https://www.greatplacetowork.com/resources/blog/how-reddit-uses-ergs-to-create-a-sense-of-belonging-at-work-q-a>

👉 [10]

https://www.researchgate.net/publication/354041646_Intimacy_Wh

37. What OpenSource Helped You Achieve?

By now I've

- 👉 *Written a heavily featured course*
- 👉 *Helped land a helicopter on Mars (According to Github)*
- 👉 *Written multiple featured articles*
- 👉 *Worked for companies in Asia, Europe, Africa and America*
- 👉 *Worked more for the Silicon Valley*
- 👉 *Spoken at great conferences even against global talk competition*
- 👉 *Mentored 100+ cool folks, including FAANG engineers in OpenSource*
- 👉 *Hit hacker news front page*
- 👉 *Organized a conference that saw nice speakers from Google, Microsoft, ElasticSearch, RedHat, the US government, CERN etc*
- 👉 *Written featured libraries*
- 👉 *...*

By the grace of the Almighty, I owe it to opensource

And folks in my own country, Mauritius, tell me they don't have time. For such a useless activity.

I mean,

- 👉 *how can you sabotage your own career?*
- 👉 *how can you limit your economic prospects?*
- 👉 *how can you throw amazing opportunities under the bus?*

👉 how can you pass up the opportunity to meet wonderful friends?

👉 how can you dunk an exciting, fulfilling developer life?

You work to achieve these 5 points, don't you?

And then, you lecture people about careers, being top achievers, impact, and social good. All while demeaning OpenSource contributors.

Some hard corporate wake-up is needed.

For my part, I am preparing to scale the pace by which I help people in tech.

38. Distributed Patterns Are Useless?!

So, Soumen Sarkar has been calling the book “Patterns of Distributed Systems” by Unmesh Joshi useless, especially the pattern part.

I think this is a gross misstatement coming from someone who has been long enough in the tech industry. Software patterns might be useless, but, I bet Soumen, you did not read the book.

First of all, this book does a wonderful job surveying how the opensource world is doing distributed programming. This survey by itself is more than useful. If you are into system design, this is one thing the author saves you from doing. It provides a foundational catalog of projects and techniques that you need to be aware of.

Secondly, the book explores the techniques used by these projects. They are not useless, Soumen, these patterns. You see a living proof of how they are used by products used by thousands of people.

Thirdly, just because academia/research points out faults in these techniques does not mean they are useless. Kafka has let's say some faults, but this does not render Kafka useless. Kafka can maybe amend those in upcoming releases. Now as for the view that we should only use vetted systems, this is a bit unwise as vetted is only until unvetted. We can only wait till the end of time to be sure that something was indeed solid.

Fourthly, as to the view that the content is good but this knowledge is incomplete, incomplete to the point of being useless. Throwing statements in the sense of: Try building something out of it or You read the book now what can you do? I'd say that for people in OpenSource, even this 'incomplete' knowledge as you call it is very useful. This book gives you an excellent toolset to x-ray OpenSource projects. When you contribute to OpenSource projects, the first step is to understand what's going on. If the project deals with the distributed, this book allows you to get an instant idea of what's going on. In the case the project developed improved techniques then, you have a sense of what and where they optimized. Even if you are reading papers. Even when you are building your own systems.

This book is an excellent resource, despite the take of cosplay trolls.

39. Donald Knuth Clarifies

🎉 At 85 years of age, Donald Knuth was kind enough to reply to my email.

In April, while writing “Merge Sort and Its Early History” (<https://lnkd.in/ecjByBbt>) , I stumbled upon the MOVEIN instruction.

I could not find it in assembly programming references.

Google returned I think few passing mentions.

Even, more weirdly, folks at asm on IRC were unaware and as confused as myself.

This was at a stage where assembly was being defined, forget about that, even the architecture of the computer, the Von Neumann architecture was not popular. Neumann was himself trying out ideas.

So, I asked Donald Knuth, the author of the paper what it meant, here is his answer.

I guess Osmar Mansoor, you were right, it was an interesting piece!

Dear Abdur-Rahmaan,

Please excuse the fact that I haven't had time to look at your email message from April until today.

In that program, MOVEIN is not an instruction; it is a symbolic name for the address where a dynamically changing instruction will be stored. (His computer was a LOT different from what we have now!)

Line 41 of the program says that we should reserve one "storage tank" (RST 1) to be called MOVEIN. Line 46 puts a PIK instruction into that storage tank. Eventually, control is transferred so that the machine executes that PIK instruction (which moves a block of words into a sequence of short tanks called BUFFER).

Thanks for your interest in history, and for your patience.

Best wishes, Don Knuth