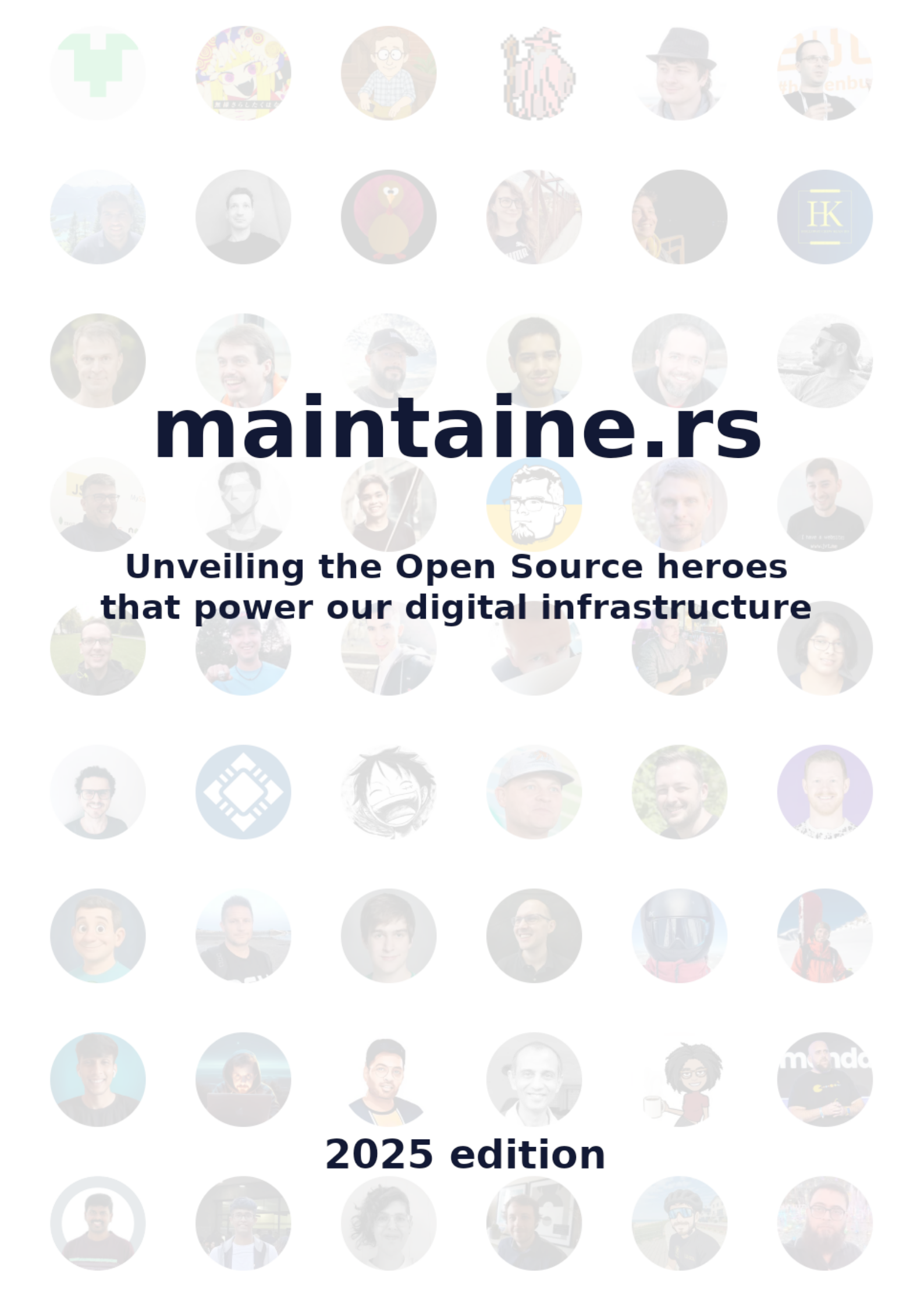


maintaine.rs

Unveiling the Open Source heroes that power our digital infrastructure

2025 edition

maintaine.rs : Unveiling the Open Source heroes that power our digital infrastructure
Edition: 2025
Editor: Nick Vidal

The **maintaine.rs** book is a curated collection of stories from top Open Source maintainers, gathered during GitHub's Maintainer Month in May 2025.

Contents

Introduction **6**

@akihirosuda – Akihiro Suda **7**

@alex – Alex Gaynor **11**

@amrdeveloper – Amr Hesham **13**

@andreashappe – Andreas Happe **15**

@atodorov – Alexander Todorov **17**

@bagder – Daniel Stenberg **22**

@blyxyas – Alejandra Gonzalez **25**

@bxcodec – Iman Tumorang **29**

@camilamaia – Camila Maia **33**

@cezaraugusto – Cezar Augusto **43**

@darccio – Dario Castañé **46**

@delta456 – Swastik Baranwal **50**

@derberg – Lukasz Gornicki **53**

@desrosj – Jonathan Desrosiers **56**

@drmohundro – David Mohundro **70**

@fabiocaccamo – Fabio Caccamo **72**

@foso – Jens Klingenberg **75**

@francescobianco – Francesco Bianco **77**

@freak4pc – Shai Mishali **80**

@hollowaykeanho – (Holloway) Chew Kean Ho **82**

@hzoo – Henry Zhu **90**

@jamietanna – Jamie Tanna **95**

@jbednar – James A. Bednar **99**

@jcubic – Jakub T. Jankiewicz **101**

@jugmac00 – Jürgen Gmach **104**

@jviotti – Juan Cruz Viotti **107**

@karlhorky – Karl Horky **111**

@karmatosed – Tammie Lister **117**

@kgodey – Kriti Godey **120**

@leandromoreira – Leandro Moreira **124**

@lrusso – Leonardo Javier Russo **126**

@martincostello – Martin Costello **128**

@mikemcquaid – Mike McQuaid **131**

@mte90 – Daniele Scasciafratte **135**

@nickytonline – Nick Taylor **137**

@niklasmerz – Niklas Merz **142**

@nolanlawson – Nolan Lawson **146**

@notmyfault – Alexander Brandes **149**

@patrickheneise – Patrick Heneise **152**

@pradumnasaraf – Pradumna Saraf **156**

@raisinten – Darshan Sen **160**

@raphael – Raphaël Simon **162**

@sabderemane – Sarah Abderemane **166**

@saikrishna321 – Sai Krishna **168**

@samdark – Alexander Makarov **172**

@sanjayaksaxena – Sanjaya Kumar Saxena **175**

@shazow – Andrey Petrov **178**

@skywinder – Petr Korolev **180**

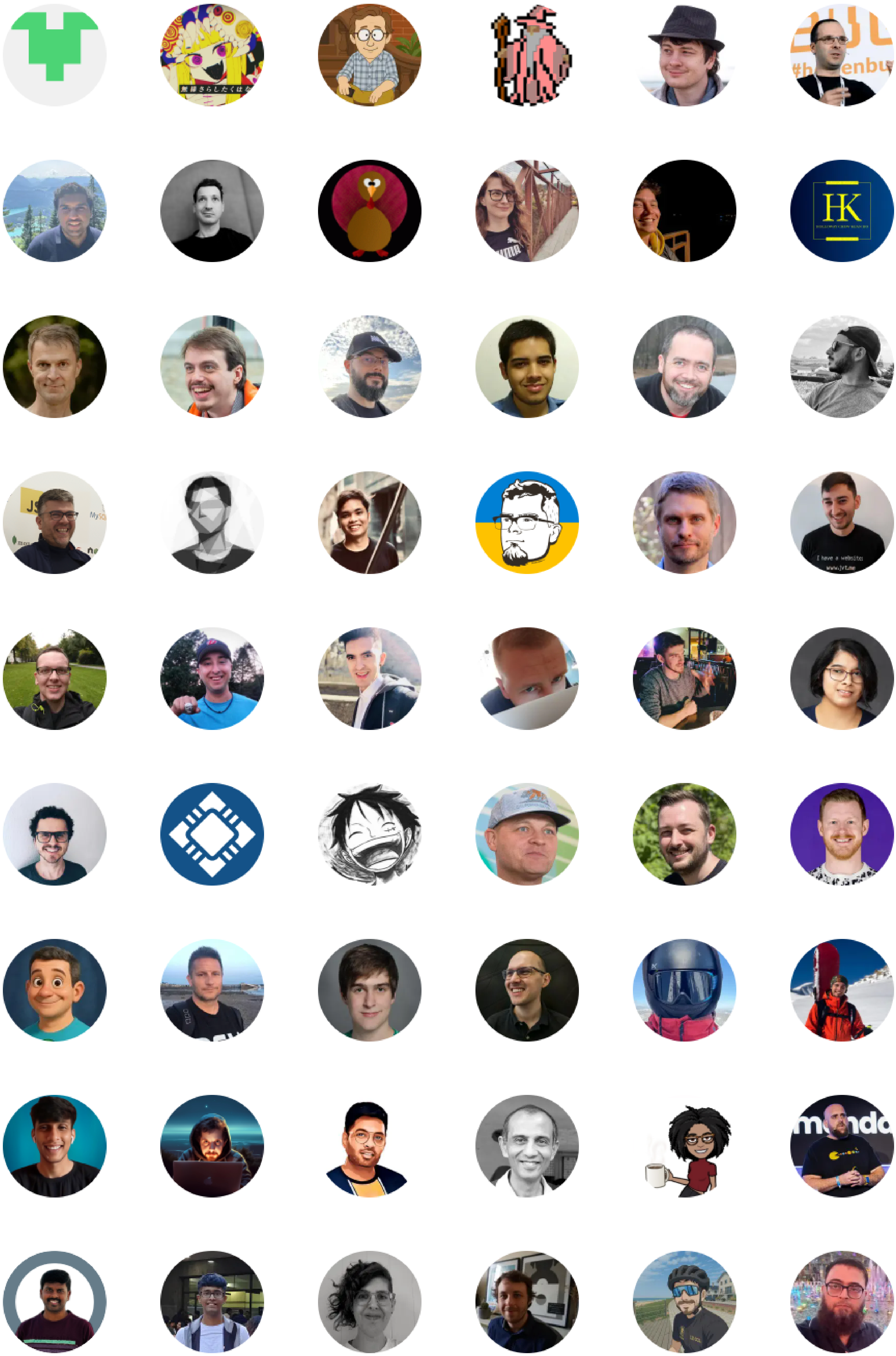
@srinivasantarget – Srinivasan Sekar **183**

@sy-records – Lu Fei **187**

@thomaspoignant – Thomas Poignant **190**

@vdemeester – Vincent Demeester **193**

@wasiqb – Wasiq Bhamla **196**



Introduction

github.com/nickvidal
maintaine.rs/nickvidal

During the month of May 2025, I've had the privilege of keeping in touch with the top maintainers across the Open Source ecosystem and reviewing their stories to share with the wider community as part of GitHub's **Maintainer Month** campaign.

Reading these stories left me feeling deeply grateful and inspired. These maintainers not only write code, review issues, and merge pull requests—they also navigate community dynamics, mentor new contributors, and increasingly adopt security best practices to protect their code and users. They literally keep the digital infrastructure up and running for the benefit of all.

From the various back-and-forth messages with these maintainers, I was struck by their kindness and dedication, which filled my heart with hope and optimism. One message from Niklas Merz stayed with me:

> After many years of doing Open Source and some periods with tiresome work, low recognition and motivation, small things like this help to relight the fire we all feel for Open Source. The opportunity to share our stories for Maintainer Month certainly had a great impact.

I hope this collection of stories brings you the same sense of appreciation and hope it brought me. Whether you're a maintainer yourself or someone who benefits from their work, may these stories help you see the people behind the code—and inspire you to celebrate and support them.

These stories are a testament to the quiet heroism at the heart of Open Source. Behind every project we rely on are people who choose to show up—not for profit or fame, but for the love of building, solving problems, and sharing solutions. Their work ripples outward, enabling innovation across industries and connecting communities across the globe. In a world often driven by short-term gains, maintainers remind us of the enduring power of generosity, collaboration, and purpose.

Let's amplify their voices! Together, we can ensure maintainers receive the recognition, support, and resources they need—not only in May, but all year long.

Nick Vidal
Community Manager
Open Source Initiative

@akihirosuda – Akihiro Suda

github.com/akihirosuda
maintaine.rs/akihirosuda

I'm Akihiro Suda, a software engineer at NTT in Japan.

I've been a contributor and a maintainer of several projects related to container virtualization on Linux for almost a decade.

Container Ecosystem I've been involved with

Docker/Moby

My container journey began a decade ago with Docker, the most popular containerization platform. While the Docker Desktop products are proprietary, most of their underlying non-GUI components have been Open Sourced (Apache License 2.0) and openly developed in the community. The Open Source parts are also known as Moby since 2017.

I began my contributions to Docker (later Moby) circa 2016 when I encountered several issues, especially those related to the filesystem. My regular commitment was well recognized, and it fortunately resulted in my appointment as a maintainer, although I had never been affiliated with Docker, Inc. I appreciate the company for making the maintainership open to the community.

In 2018, I implemented the Rootless mode as a major functional contribution, which strengthens security by running the Docker daemon without root privileges, leveraging technologies incubated in LXC and runc, and my new user-mode networking stack (slirp4netns and RootlessKit). Rootless mode was upstreamed into Docker in 2019. Ahead of Docker, some portions of my work were also incorporated into Podman, an alternative implementation of Docker by Red Hat, which had been also pursuing Rootless containers at the same time. I was pleased to influence both projects.

BuildKit

BuildKit is the framework used by the modern implementation of the `docker build` command for building container images efficiently.

I was appointed one of the initial maintainers of the project in 2017, as I was already proposing a similar (but much poorer) mechanism on my own. My own work was just untidily grafted into the legacy implementation of `docker build` and quite inferior in extensibility and maintainability.

BuildKit was established to rethink the whole design of `docker build` from scratch. Through the collaboration in the community, the project successfully enabled innovations such as concurrent task scheduling, efficient caching, and an extensible Dockerfile format.

containerd & runc

containerd and runc are the common runtimes used by both Docker and Kubernetes.

containerd provides high-level gRPC services for managing the lifecycle of containers and container images. runc provides low-level wrappers for Linux kernel's facilities such as `namespaces(7)` and `cgroups(7)`, following the Open Container Initiative (OCI - not to be confused with "Oracle Cloud Infrastructure") Runtime Specification.

I've also been a maintainer of containerd (2017-), runc (2020-), and OCI Runtime Spec (2022-). It may sound like I'm maintaining so many projects, but it's just a small world: all these projects are tightly interwoven in one ecosystem, and a change in one project often incurs changes in other ones. So, it is crucial to coordinate these projects closely.

Coordination is tough, though; it is quite common to take multiple months, or even years, to implement a single feature. A feature proposal is sometimes stalled due to fierce objections - but this case is rare. The actual reason is often due to bikeshedding, or simply due to forgetting. It still remains an open question how to advance a proposal that did not get much attention despite its usefulness.

nerdctl & Lima

In 2020, I launched nerdctl (contaiNERD CTL), a Docker-compatible CLI for containerd, so as to facilitate experimenting with the cutting-edge features of containerd that were not present, and could not be easily implemented, in Docker at that time.

In the following year I also launched Lima (LInux MAchines), a tool to create a virtual machine optimized for running containerd and nerdctl. I originally designed Lima as "containerd Machine" to bring the concept of the former Docker Machine into the containerd ecosystem, but I changed my mind after all and released Lima with the leeway to allow non-containerd workloads too.

Both projects were well received in the community, and adopted by several third-party projects such as Rancher Desktop (SUSE), Finch (AWS), and Colima.

Through launching the two successful projects, I realized again the importance of the people. The key to success is how to attract contributors, and how to keep them motivated. This includes showing appreciation, sorting out "low-hanging fruit" tasks, accommodating release schedules, and maintaining clear governance.

My Journey Onward

I'll still continue to work with containers; however, I envision that containers in the next decade may not look like that of today.

Stronger Isolation Technology

With the rise of LLMs, people now have a growing tendency to execute arbitrary code without reviewing them at all. Contemporary containers are still effective in alleviating the security risk in executing malicious code; however, sophisticated malware may break out of the container by exploiting vulnerabilities of the container runtime or the Linux kernel. The next decade may see the drastic revival of virtual machines (e.g., Kata, notably with PVM) to strengthen the isolation, at the cost of performance, increased memory footprint, and reduced flexibility. However, this does not mean that the current container ecosystem, and the community, have to be scrapped. Even when a technological trend shifts, the community still remains there to help maintain interoperability across the old and the new stacks.

WebAssembly

WebAssembly (WASM) is also emerging as a secure alternative to containers. It is also promising for offloading server-side logic to web browsers.

However, migrating a container image to WASM is not always easy due to the difference in the toolchains.

My teammate Masashi Yoshimura is now working on elfconv project that directly converts ELF binaries in a container image to WASM. This is quite a challenging project, as an enormous number of CPU instructions and Linux syscalls have to be reimplemented for WASM. Our success will depend on whether we can build the community for collaboration on reimplementing them.

Library Sandboxing

When writing software, it is practically inevitable to depend on third-party libraries. This supply chain is now under attack. Notably, the liblzma incident in 2024 demonstrated that even

well-known libraries could be compromised by a malicious contributor. Also, there has been an ongoing incident of fake Go modules published with very plausible content and a high number of GitHub stars.

To mitigate such attacks, I'm now working on a new project "gomodjail", the "jail" for Go modules. gomodjail is similar to containers in the sense of syscall restrictions, but it works in much finer granularity. My current focus is on the Go language, but I hope that I will be able to apply the same sandboxing technique to other languages too, as similar incidents have occurred in other language communities.

@alex – Alex Gaynor

github.com/alex
maintaine.rs/alex

My name is Alex, and I've been involved in Open Source software for more than 15 years. My first contribution was to Django, the Python web framework. Every day, I'd go back to Trac to see if there were any new replies to my issue and patch, and over time I started to see other issues that I could help out on (even if it was just to point out that one issue was a duplicate of another). My contributions snowballed from there. These days, while I work on many projects, the ones that get most of my time and attention are the Python Cryptographic Authority family of libraries, which are the most popular cryptography libraries for Python.

The impact of AI on Open Source

The easiest prediction is that the impact on Open Source will be similar to the impact on software engineering in general. While this is surely true, it's not terribly interesting. Instead, I want to share two experiences I've had, and one I'm hoping to have.

The first experience I've had is users submitting low quality pull requests that strongly appear to be the output of a not-particularly-good LLM. These are frustrating and a waste of time. I don't care if a PR was developed with an LLM or not, I care whether it's low quality. And one thing you can certainly do with an LLM is produce large amounts of low quality code and accompanying prose, forcing maintainers to wade through plausible sounding nonsense before rejecting a PR.

A more positive experience I've had is using an LLM to write a PR. Our project gets a fair number of feature requests which would be entirely reasonable for us to add, but for which I have absolutely no motivation to work on. I'd be happy to review a PR, but not enthused to do the work myself. By using an LLM, I can square this circle: the LLM can generate a patch (including docs and tests) and then I can review it. This is not a solution to a technical problem – these feature requests are rarely especially complex, and indeed this workflow only works *because* I'm entirely capable of writing the code myself. Instead, it solves a motivation problem: some feature requests I never find myself enthusiastic about, but I'm always in the mood to review code.

A final experience, which I have not had yet, but which I'm hoping LLMs can assist with is issue triage. A substantial portion of the issues filed against us are installation issues, almost all of which involve a misconfigured Python environment. While I have a lot of sympathy for users who are confused and frustrated by Python's packaging tools, these issues are also exhausting for maintainers. I'm daring to dream of a world where maintainers can mark issues as appearing to fit a particular archetype, and allow an LLM to provide guidance to a user to resolve their issues, or alternatively to gather enough information to identify that something might really be a bug. All without requiring a maintainer to exhibit superhuman patience.

@amrdeveloper – Amr Hesham

github.com/amrdeveloper
maintaine.rs/amrdeveloper

I'm Amr Hesham (Known with username @AmrDeveloper), and I am working as a Compiler engineer, and today I'm writing about my way into Open Source..

Currently, I am the creator and maintainer of GitQL, LLQL, ClangQL, LinkHub, TurtleGraphics and some other nice projects, and I contribute almost daily to the LLVM foundation, especially in Clang, which is a C/C++ compiler and in the past, I contributed to other projects like Rust analyzer, CheckStyle, etc..

How did you get involved with Open Source?

My journey with Open Source started back in 2018 when I was learning computer science subjects on my own, and on each course project, I used to just take it one step forward to implement more features, document features and share them on Github.

After a while, I was searching for an Android UI library to create a reaction button similar to social media, for example, like LinkedIn and Facebook, to provide emojis and dialog once clicking on the like button, and then I found that many Android Developers have the same use case so i decided to design and implement my first library to cover this use case but with a lot of customization options and released as an Open Source library, i surprised by the good feedback, contributors, people send messages about how they used it in their project.

After that, I implemented more libraries and tools to solve use cases for Android developers, and I learned a lot of technical and non-technical skills during that time, but I found that I can continue experimenting and improving my libraries or ..... I can contribute to other projects and learn from senior engineers who have built libraries for a long time, so I can contribute, learn and then contribute back to the community, then repeat the circle :D.

Now, 6 years forward, I am still doing the same circle every day, not only contributing to popular Compilers and tools but also creating my own languages and database engines that are used by around 100k users.

What's Open Source to you?

For me, Open Source has a lot of benefits for everyone, but I would like to highlight one important part, which is providing an amazing way to learn and practice in real projects that are used by millions, maybe even billions of people, in a safe community with mentors who share their experience in a good way, for example whatever the subject that you are interested in you can take course or read a book about it thats essential but in my opinion that the best return on investment is to try to contribute in real project in that area and be a part of the community.

What are the main challenges you face as a maintainer?

Maintaining an Open Source project is interesting, but it also comes with some challenges, for example, managing time between your full-time job, your Open Source time and family time is tricky :D, also maintaining your motivation, managing the community and make it safe for everyone to share ideas and contribute and mentor new contributors.

What are some ways contributors can better support maintainers?

There are many ways to support the maintainers, such as contributing to the source code, documentation or in the community channels like Discord, also sharing ideas, reporting issues, reviewing code and sponsoring the project if you can.

What's the impact of AI on Open Source development?

In my opinion, AI tools can help you to start contributing to the project in a faster way, for example, building and understanding parts of the project, so you can use it as another source of information, and you can also use LLM to get a quick code review locally before submitting your code.

What advice would you give to current and new maintainers?

It's essential to maintain your motivation and organize your time, enjoying what you do and always remember why you started to create or maintain the project.

@andreashappe – Andreas Happe

github.com/andreashappe
maintaine.rs/andreashappe

Hi, I am Andreas, living in Vienna, Austria. I'm a developer gone penetration-tester gone PhD student focusing on how to use LLMs for offensive security. People tell me, that I breathe security.

What Projects are you currently involved in?

Currently I work on hackingBuddyGPT and cochise: both are research projects that use LLMs to hack real systems. These projects enable security practitioners to experiment with using LLMs in very few lines of code. In the long term, I believe that LLMs will democratize access to security testing.

I am also one of the leaders for two OWASP projects: the Top 10 Proactive Controls and the upcoming OT Top 10. The Proactive Top 10s describe common security measures and techniques that software developers should be aware of. I am not happy that we typically only highlight how to attack systems and not show how to better protect ourselves.

The OT Top 10 are currently under development. OT stands for Operational Technology, typically this is technology that controls some sort of physical process. Think factory floors, power networks, etc. We need to improve the security of these critical infrastructure systems.

Why Open Source?

I don't understand the question.

I was learning to code in the late 90s, living in Austria's country-side. People were talking about this new thing, Linux, and I became intrigued. Online, I found my tribe and have been a proponent of FOSS since. David Roe's awesome firstcommit.js gives my first commit as a documentation/configuration fix for the linux kernel in 2000 (although I believe that this commit happened around 2003/2004).

How do I feel about Open Source? How does a fish feel about water? Open Source and its community has been a big influence on my life. This is true for most of us, some of us might just not know that yet.

I work in the security domain. In my opinion, Open Source security (and FOSS security tooling) raises the collective security for all of us.

What do you think are the biggest security challenges facing Open Source today?

I see two two challenges:

1. Maintaining Trust

   Sophisticated attacks against OSS maintainers erode the trust that is fundamental for online collaboration. Collaboration is the life blood of OSS, so we have to take this threat very seriously. If you think about it, other hyped security problems such as Supply-Chain Attacks (attacks against your dependencies) also boil down to trusting your dependencies' maintainers. Getting to know them (including face-to-face contact) is thus becoming more and more important.

2. Making Security invisible and unobtrusive

   I believe in making it easy to do the right thing, especially when it comes to security. Developers should not have to explicitly think about how to make things secure, the "normal" way of solving a problem should already be the secure way. Frameworks that offer sane secure defaults have had a great positive impact on web application security.

And a bonus one: Don't use Security to Shame People

Typically people can only keep a few things on their mind (3-4 things). Imagine, you're a developer. Those four things are quickly used up: one or two functional requirements, performance, food & coffe, and maybe there's something going on in your personal life. Now, imagine a pen-tester gets to work on the same projects. What's on their mind? security, security, security, coffee. Of course, they will find issues.. and that's okay. We're here to help. An audit is just an opportunity to learn and fix problems before real attackers find them.

What's the impact of AI on Open Source development?

It's gonna be an interesting ride. On one hand, I believe that AI will allow many more people to create code. This is a big enabler.

But once there's the code, you have to maintain it and keep it secure. For this, you need a deep level of understanding of how code works and be social with your contributors. "Just" depending upon AI and ignoring both education and social dynamics will not be the perfect solution in the long-term.

@atodorov – Alexander Todorov

github.com/atodorov
maintaine.rs/atodorov

My name is Alex, a software quality engineer and fellow Open Source contributor from Bulgaria. At present I am a co-organizer of FOSDEM's Testing and Continuous Delivery devroom and the project lead for Kiwi TCMS - a popular test management system.

You can follow what I do at https://github.com/atodorov/ and https://kiwitcms.org

How did you get involved with Open Source?

I have been involved with Open Source for a very long time, almost 30 years at this point, however in the early days I was only a user. And not a very skilled one at that. During my early teenage years I became interested in computers and programming and somehow started experimenting with FreeBSD and early versions of Red Hat Linux. It was probably more of a coincidence than anything else.

Much later, early 2000s if I remember correctly, while working on some small project I remember having issues with one particular Perl module which would inspect image files and return their width and height. It didn't work correctly for the latest version of the Flash file format. Everything being plain text I was able to make some adjustments and make it work on my computer. At some point I realized I could just email the author of this module and tell them how to improve the code. Which is precisely what I did, very naively I should add, I didn't send a patch or anything like that. I just described which lines in a function need changing and how. That was my first contribution to Open Source.

What's Open Source to you?

Nothing less than everything, really. Open Source is heavily infused into my personal and professional life for the past 20 years. As fate would have it I haven't really worked much outside of the Open Source industry.

Shortly after that initial contribution I started my first paid job as a software developer and it was only natural to be looking for job opportunities with Linux as it was my primary operating system for home use.

I was lucky I guess. I was writing software in Pascal on Linux which was exactly what I wanted. Meanwhile I started contributing translations and small bits into various projects, not even sure why. I think I was just experimenting at that point.

Shortly afterwards I got lucky again because my passion for Open Source was recognized by Ben Levenson at Red Hat and I accidentally landed my dream job. I took a leap of faith and relocated to another country to become a software test engineer. All this just because I wanted to work with Linux.

This is where I noticed first hand how Open Source is actually put together and started contributing more heavily with technical contributions. I recall one of my more meaningful contributions was in virt-manager, GUI improvements, in 2007. Then it gradually escalated around the software tools and components I was using at work.

What projects are you involved in?

Currently I am working on and off on several Django packages which I use as dependencies but primarily on Kiwi TCMS which is a popular test management system and arguably one of the very few Open Source variants left standing. It is a web based application used by QA engineers and their managers to track and organize testing related work. Think of it as one of the many pieces which you have in your software development and delivery infrastructure. Such systems are also part of what's called Application Lifecycle Management and could be found in large organizations.

Kiwi TCMS itself has a classic Open Source history - it was created by Red Hat, then published on GitHub, then discontinued and left to slowly die out. I forked over after most upstream activity had stopped. In fact forked three different times until it became Kiwi TCMS as we know it today.

How do you grow your community?

Hard question. In the early days I was more centered around finding volunteers, working with students (for example with Major League Hacking) and doing coding sprints at various conferences. These days my focus is more towards building a sustainable Open Source business practice and being able to support a core team of maintainers than anything else.

What are the main challenges you face as a maintainer?

As in countless other projects our core user base doesn't necessarily have the skills to contribute technically and most of the lower hanging fruits have already been picked up. The challenge of

course is being able to respond to feature requests and free (as in beer) support calls for users with less technical experience.

There is also the ongoing task of maintenance and just keeping up-to-date with new dependency versions even without adding new features in the application itself. Lately it's also become apparent that I need to become even more involved with some of our dependency stack and help out other fellow maintainers.

What are some ways contributors can better support maintainers?

Here's a broader answer which also includes consumers of Open Source:

Think about you human fellows who spend their time and often sacrifice a lot just to deliver the library that you need. This is especially true for small communities.

Don't demand anything from the maintainer, don't be rude, be patient and try to help yourself before engaging with the rest of the community.

If you are using a project commercially, consider helping out. Other fellow developers will value your technical contributions and your effort to learn more than anything else. Ask for advice or pointers, hack up a proof of concept and be ready to iterate as many times as requested on the schedule of the maintainer. Plan for this and be a good Open Source citizen.

If you can only offer monetary support, ask your company to do so adequately or better yet seek commercial support where possible. $50 for this new feature or a bug fix that you depend on is probably not going to cut it.

What are some of the key security practices you've implemented in your project?

I love this question because originally our code base did have lots of vulnerabilities. The most critical one was remote code execution which was relatively easy to exploit.

Being a tester myself I view security as just another piece of the software development puzzle that we need to integrate with our day-to-day workflow. What I like to do on most projects is:

- Enable all kinds of linters and static analysis tools that are relevant. This helps me clean-up code smells and security related issues which are due to common mistakes.
- Enable all tools and keep enabling new ones over time as I get to know about them. Disable the ones which don't appear to bring value. My current favorite is MegaLinter, although there may be more suitable tools depending on the programming language used.
- Scan all of your code base plus the entire dependency stack and take action as soon as you can. When I say "all" I really mean all of it - software, its test suite, infrastructure as code,

CI scripts and YAML files, etc. It is astonishing what you would find and at the same time it is a mountain of work to sift through all the reports and patch out (or just silence) issues further down the stack. Most often than not that means asking other Open Source projects to adopt similar tools and practices as you do which opens more and more work for everyone involved
- Use tools to automatically upgrade your dependencies to their latest versions
- Establish a security policy and channels for researchers to disclose issues responsibly. Kiwi TCMS used to have several channels for doing this, currently it's only via GitHub
- Join security bounty programs if applicable - Kiwi TCMS has been part of one in the past which helped discover and fix vulnerabilities on our side
- If you are running a production instance, for example a demo installation, enable traceback reporting, e.g. Sentry, which helps you find out various kinds of failures you had never imagined. I tend to view these as critical and possibly security related until I get to know otherwise

One thing which I am interested in doing but not quite there yet is to engage with some sort of penetration testing service / security audit service, preferably led by a team with a strong affinity towards Open Source. However this is something which doesn't apply to all projects.

From the point of view of your own project all of the above means constant maintenance and very likely refactoring and updating your implementation as you get to know things. Since Kiwi TCMS started as a legacy code base I've worked on extensive refactoring in several key areas. Some areas have been refactored two or three times until they got to a point which is elegant, easy to maintain and relatively secure.

What do you think are the biggest security challenges facing Open Source today?

I'm not quite sure. It's probably a mix of lack of development time/resource; lack of some technical knowledge about what may constitute a future problem; lack of overall vision towards security and probably a healthy dose of "I don't care".

That's also a challenge on the side of the consumers of Open Source. They need to be as equally aware as the developers they trust. Just because something is Open Source doesn't mean it is secure.

I've seen plenty of blatant violations with some of Kiwi TCMS' own forks. For example people publishing their SSH keys or account passwords on GitHub and not changing them even after I send them an email to alert them about their mistake. In one particular example I've seen this continue for years.

Mistakes do happen but let's learn and improve when we discover them.

What's the impact of AI on Open Source development?

I can't say. I haven't used many AI tools myself and I am a bit skeptical that they provide good value. Maybe I'm just old school.

What advice would you give to current and new maintainers?

Being a maintainer is oftentimes grunt work which takes its toll and one should really love what they are doing to continue doing it. Oftentimes it looks completely irrational "working" as a maintainer, especially on smaller projects. Had I known how much work and challenges there would be being involved as an Open Source maintainer and contributor I would have probably not done it!

To anyone new I would say: spend enough time to find out what you are getting yourself into and make sure that this is what you really want (for whatever reasons you may have)!

@bagder – Daniel Stenberg

github.com/bagder
maintaine.rs/bagder

I worked full-time on a project that was just one hundred lines of code in 1996.

In the early 1990s a friend of mine helped me land a "computer gig" at just twenty years old. I had not attended university, but I had already spent years wasting all my spare time programming my C64 home computer in assembler, writing demos and games.

This particular gig was for a company called IBM. I was to install and customize their RS/6000 unix machines for Swedish customers. The hardware arrived at this location where each machine would be installed and personalized according to the customers' orders and wishes.

A colleague at this new job would soon proudly show off his 1/4" tape "full of source code". Lots of source code for programs, completely free! Most of them were written in C, a language I had been dabbling with for a few years already by then and this took me out to deeper waters.

A few years later I was the co-author of an IRC bot - released in the open for everyone and I had a lot of fun with that. The code was made freely available but not "Open Source", because at that time nobody had yet heard the term.

A hundred lines of code

As a side thing for the bot project, I got involved in creating a small tool for downloading content over HTTP in late 1996, and it did not take long until the side thing became my main spare time project. In the spring of 1998 we renamed that project to curl and a special journey had started for real. Oh, and by that time the term Open Source was also coined so now we had a term for what we were doing!

Those projects of mine were Open Source from day one as I wanted to contribute to the ever growing collection of free and useful code out there. I was acutely aware that I needed help from others to make my projects polished and to get them to properly work in many different environments - to reach success basically. I asked for people's bug reports, patches, feedback and comments. In the beginning they were few and infrequent, but the more the project matured and improved, interaction with others increased. I have always tried to thank contributors and give them proper credit, as that is often the only currency available. I always try to be inclusive

in wording and speak of “we” and not “I” about the project, even in times when most of the changes were done by me.

We shipped new curl releases early and often. Added features, fixed bugs, iterated. I spent much of my spare time on this for years to come.

Later, through the decades, I would co-start more Open Source projects (like Rockbox) and I would “take over” and push development forward in others (libssh2, c-ares) but my original main guiding principles remained. Respect contributors, remain inclusive, lower contribution friction as much as possible, give credits. Of course in addition to writing excellent code that is documented clearly so that people get a better chance to use the products and help make them better.

To Mars

In the early 2020s, when I had started to finally work full-time on curl and when we estimated curl being used in somewhere around twenty billion installations we learned it was involved in the helicopter landing on Mars. A surreal milestone.

The spare-time-approach to curl from the 1990s is quite different from the one-of-the-most-used-software-components-in-the-world-approach we have today. These days we follow procedures and protocols. We tighten every bolt iteratively. We do reviews, run scanners, analyzers and fuzzers, we sign tags, commits and releases and we do fully reproducible release tarballs etc. We spend significant efforts on security and thinking about how to further improve security.

We have grown the project to serve as a digital infrastructure cornerstone with excellence. We strive at being top of the class in every aspect when it comes to Open Source, code quality, documentation and security. This, while at the same time driving development forward at a high speed to offer the world the Internet transfers they need, want and deserve. Give something a lot of time and you can really accomplish something.

More, better, brighter

The curl project is of course not free of challenges and they are probably not that different from other projects of similar kinds. We are a handful of aging maintainers that are all (too) similar to each other: old western white men. We are always looking for more maintainers and more diverse contributors. We run an infrastructure project that is mostly invisible and out of people’s minds and we struggle with the financial side to ensure that curl can live to be a hundred years old.

Open Source use and development have truly exploded since the term was born. It is now used everywhere but we are not done and there are still many things to work on. We need to make it more sustainable, with maintainers and funding. We need to keep improving security in source code, infrastructure, tooling and supply chains. We need to take care of the maintainers we have. If you can't complete what you want today, just continue tomorrow or the day after. No stress. Do it for the fun of it. In the great scheme of things, no one will care if you did not complete it yesterday.

Let's make an awesome Open Source future.

/ Daniel Stenberg, April 28, 2025

@blyxyas – Alejandra Gonzalez

github.com/blyxyas
maintaine.rs/blyxyas

I'm Alejandra, one of the people that maintains Clippy, Rust's official linter. And for this Maintainer Month of May I've come to opensource.org about some often-overlook aspects of maintaining a FOSS project, some of my personal story with FOSS, some tips about software security, and how to better help maintainers as a contributor. Strap in because this will be a wild ride!

Who are you again?

As I said, I'm one of the people that maintains Rust's official linter, Clippy. You can execute Rust's linter any time[1] if you have [Cargo] installed via `cargo clippy`. I've been working on Clippy full time as a maintainer for about 2 years. In that time I've implemented several lints, fixed a lot of bugs, reviewed hundreds of pull requests, implemented benchmarking tools into Clippy and integrated Clippy into other benchmarking tools. Even proposed a Rust Project Goal that got accepted!

While I'm not the oldest maintainer, (not even close) I have some things to say, and I think that my advice could be valuable to whoever is happy to hear it.

The hard aspects of being a maintainer

While being a maintainer is wonderful (that's why I do it), it takes a special kind of person to revisit the same software every day for years without a monetary driving force behind it.

For me, some hard aspects include:

- Having a Life/Work/Open-Source balance
- Maintaining even when you have outside conflicts.
- Taking part and giving feedback on the schedule that the contributors deserve.
- Making hard decisions, taking everything always into account because ultimately you're the final say into a lot of things

[1] (unless you have a `minimal` profile set in your configuration).

- Avoiding burnout (this is a really important point and should be talked about more)

I've struggled with all these aspects in different stages of my life, reaching some kind of stability is always hard because the environment is always changing (both in and out of FOSS).

At the end of the day we're humans managing human work. With people that deserve their reviews, reviews that deserve their quality assurance (in a timely manner) and users that deserve that their issues be resolved. Everything takes a little bit of time, and we only have limited time in our day.

You will eventually learn to balance everything out (not that I've completely reached out that priced phase yet). I promise ;)

What about security?

Security was this Maintainer Month's topic, so I'll also give out some pieces of advice from the bottom of my heart. I'm trying to avoid those blanked, safe statements like "make sure you have good tests" because they don't really help anyone. I'll give more concrete tips at the risk of sounding too specialized.

1. Always keep in mind where your code will be run! It's easy to forget that your code will run on all kinds of architectures, on all kinds of operative systems, sometimes in a server, sometimes with arbitrary user-controlled inputs. This includes but is not limited to:

   - Arbitrary Unicode inputs.
   - Strings Arbitrarily-length
   - Maybe URLs or paths to files
   - File system access.

   If there's **any** probability that an end-user might exploit your FOSS project to get access into someone's server, you should disclose it!

2. Keep your CI pipelines safe

   - You probably use CI (and if you don't, absolutely do!) as a way to test your project before launching it to the greater product, make sure that your workflow files are safe! Don't use unknown dependencies (in fact, use as little dependencies as possible), with as little external applications as possible.
   - Each dependency on your CI pipeline (this includes applications / bots in your repo) is a possible vector of attack, each `run` field is a weak point.

3. Better and smaller pull requests produce better code

- Avoid big pull requests. Making pull requests smaller is the best strategy to improve review times, code quality and overall team mentality.
- Pull requests under 150 lines are reviewed the fastest and thus, can fix issues the fastest.
- As a general guideline, always think about all the boundaries that the pull request code might handle, and how to break in the worst way possible that poor contributor's code.
- Get in the mud, explore the deep end of your contributor's code[2]. Break your tests, read documentation for every single one of the added functions, see if functions could be removed, check if loops could be early-returned.
- Don't forget to talk to your contributors about documentation!

4. Use automated tooling

   - This is a very simple step, don't guess about memory leaks, use a heap memory profiler (like heaptrack). Don't guess about memory safety, use a static code analyzer or a language that has memory safety built-in (like Rust). Don't guess about the origin of something, bisect it in your program (like with `git bisect`).
   - Know your system, the better you know the tools you're using the better code you'll produce and the faster you'll be able to iterate on a design. With this I don't mean learning a fancy-schmancy IDE or keyboard layout, but learning to make `perf` valuable, learning to read stack traces, learning to efficiently search throughout your documentation to find that edge case that has been bugging you out all week.

5. Keep learning

   - Even if your brain is huge, with the best-quality gray matter in this sector of the galaxy, there have been people before you. The big advantage we have over humans before our time is the collective knowledge that aids us to achieve excellence. Do not let your ego and pride get in the way of making great software, because precisely the way to make great software is knowing who to ask the right question to get the desired results.

What can I do as a contributor?

First, thanks a lot for wanting to help your fellow developers! The biggest help that you can provide (for me) is not really in the code itself, but in what surrounds that code.

[2] You can actually fetch a remote pull request with `git fetch origin pull/$pr_num/head:$branch_name`!

- Improve your pull request descriptions, have detailed commit messages (with descriptions directly on the commits!), split your pull requests into several commits and for features, don't try to cram too much into a single pull request. Review you yourself your pull requests (in the Pull Request interface itself) before publishing it, you'll find a lot of slip-ups this way.
- Help with long-standing issues, if you're a trusted contributor, help to give labels (if you have that authority). Give your opinion, get minimal reproducible examples of buggy behaviors!
- Help with documentation! Not everything has to be about code, adding and updating documentation is probably one of the most valuable things that you can do in a codebase.

Conclusion

And that's everything I'll talk about today! I hope that you have learned something new and used your critical thinking skills to decide if these thoughts fit your mental model. Thanks for listening to my ramblings and have a great week. Peace!

@bxcodec – Iman Tumorang

github.com/bxcodec
maintaine.rs/bxcodec

I'm Iman Tumorang, currently working as a full-time engineer at Veriff, one of the biggest ID verification solution startups in Estonia. Prior to this, I also worked in a couple of industries, such as payment, media, and CRM. Throughout my professional journey, I've been following the open-source community. Even though I haven't made considerable contributions to any popular projects, I can proudly say I've been maintaining a couple of my Golang libraries, which a few people fortunately use.

I noticed a correlation between my professional career and open-source contributions. All my Golang libraries are inspired by the problems I encounter in my work. Sometimes, I need to create a small library to speed up development time. If I see that it can be helpful in the public, I will make it open-source; if not, it will remain an internal source within the company.

In this story, I will share how I delved into the world of Open Source, including the learnings, the challenges, and, of course, the fun of it.

. . . How did it start?

Back in 2016, my first job, I was lucky enough to land a position at a small startup in Jakarta, Indonesia. It was a company that offered CRM solutions. However, perhaps because the market was a bit unstable at that time, the company shut down its business six months after I joined. One month before the company publicly announced its shutdown, the CTO had already advised us to look for new jobs. As a junior engineer, it was challenging. What would I say to a new interviewer if they saw my resume showing only six months of experience?

So, what did I do? In one month, I learned what the new industry trend is. I saw a couple of big companies (startups) moving to Golang. I knew this because I have a couple of friends working at some of those big companies. So, I learned Golang, even though my stack initially included only NodeJS and Ruby on Rails. I dedicated my time to learning Golang while looking for new opportunities.

I made my first Golang library, https://github.com/bxcodec/saint. Back then, I didn't clearly know anything about Open Source; I realized that later. I built that simple library just for

the sake of my portfolio and also as proof to recruiters: “Look, my resume shows I worked with NodeJs and Ruby on Rails, but I can learn Golang in 1 month, including building a new open-source library that people can use.”

Luckily enough, with some friends’ connections, I landed a new company that will become my starting point for creating more libraries and getting noticed by many companies that actually affect my career.

Proudest Moment!

So, in my second company, I learned a lot, especially about engineering practice and Open Source. Thanks to my former Engineering Manager—shout out to: https://github.com/uudashr. I still remember when I was working with Go, using VSCode. I saw my manager’s library being used as one of the official libraries for the Golang plugin in VSCode. Wow! How amazing was that? Almost all Go engineers who use VSCode rely on his library. When I saw that, I was so amazed by him.

Learning from him, I started to make some blog posts too. I became a “Curious Engineer” who is always learning new things and creating new things. One of my posts became popular overnight, suddenly, out of nowhere, including my GitHub repository, gaining popularity: https://github.com/bxcodec/go-clean-arch. That’s when I felt I had achieved something I never expected. I feel so proud that everyone is looking at my code and even using it as a reference. In real life, when I attend Golang community meetups, people talk about the code I made; they praise it and even tell me that they use it as a standard in their company. This is what it feels like to make an impact, even without really working in their companies.

Not stopping there, I made another blog post and a new library in Golang, https://github.com/bxcodec/faker. It is also popular; I can see some people are using it. The statistics (download/clone count) show that they’re using it in their projects. When it always gets downloaded/cloned, it’s probably because of the CI/CD. Once again, I feel it really makes me happy that I can be impactful on the community.

And because of these two projects on GitHub, I became one of the top Go GitHub developers in Indonesia, LOL. It is based on GitHub stars gained.

With these stats, I can easily apply to any company in Indonesia back then. At least I will get noticed first. Even though I know it’s just a kind of fun thing to do, I think it helps motivate me to work more on my open-source libraries. I address the new issues that people raise in the library, or maybe think of creating a new library that might be useful for others.

Challenges as Maintainers!

Since then, I've created a couple of libraries, but I think because I was not that active anymore (fewer blog posts), my newer libraries are not gaining any traction like my faker library. In the meantime, I was also getting busier at my job, solving business problems, dealing with life, COVID depressions, etc. It was challenging to stay optimistic. This also affected my libraries; I somehow wasn't able to keep them maintained anymore. I will only take a look when it's a critical issue (e.g., security, breaking changes, etc.). For small issues, usually there are always good people who make contributions and fix them for me. So I just need to review and approve them.

One of the biggest challenges is prioritizing. For example, I have a full-time job, which involves dealing with business issues and sometimes tight deadlines. It can be stressful and frustrating. These conditions only lead me to deprioritize the libraries that I maintain. I felt guilty because I sometimes see people raising issues, yet I never reply until five months later. This has become a problem. For some important libraries used by many people, such as https://github.com/bxcodec/faker, I moved them to a new GitHub organization, where I can invite other contributors to help me maintain them. However, for libraries that are not widely used, I maintain them myself.

I remembered that GitHub released a new feature about donations, but still, my projects are not significant enough to make people willing to donate. It's just a small library that can be easily replaced by others. So donations are really not helping me to stay motivated; I can't switch careers to be a full-time open-source maintainer. Maintaining Open Source is a noble responsibility; you must not expect anything in return. When people donate, don't take it for granted; instead, be more serious about maintaining the project.

To address this motivation or prioritization issue, I actually made my library used by the company I worked for. For example, this library https://github.com/bxcodec/dbresolver is about managing DB connections for both RW and RO connections with load balancer functionality. I created this while working at Xendit, the biggest payment gateway in Southeast Asia. This library is used in one of the core projects at Xendit, which gives me some responsibility to maintain the open-source version since I have a real user utilizing it. Additionally, I can leverage it for marketing by saying, 'Hey, this company is using this' – even though I was the one who developed it, LOL.

Final Thoughts

I'll be real, I'm not a noble person. I worked for money to support my family and my lifestyle. Working for free is not my style. Working on Open Source definitely does not give me any

monetary value directly. Instead, what I gain is a better portfolio and a new network through collaborations that can later help me land new jobs through referrals, etc. For most of the biggest players, yes, they can get donations, but for small fry like myself, I don't dare to dream that much. I believe that helping people with a spirit of community will benefit me later in different forms. It doesn't have to involve money every time.

For the community itself, Open Source helps us advance technology. With many people collaborating on one project, it helps us shape a better approach to building systems. People can easily look at the source code and make improvements if they think they can make it better. I've experienced this many times; thanks to all my contributors, it helps me think and shape my knowledge about building an extensible, reusable, and scalable library.

Even though I was really busy with other stuff in my job, I always tried to spare a couple of hours each month to take a look at my Open Source projects. Sometimes, if someone raised a PR, I still tried to prioritize it during the weekend. However, it is only possible when there are no urgent matters like family, job, etc.

Lastly, you're doing well for all the maintainers who read this! When you feel down, it's okay to take a break, as there are many important things in real life for you. Take a short break and return with recharged energy. There's always the option to extend responsibilities to new contributors; inviting new people to become main contributors could be a possibility. Unless you receive regular donations that cover your monthly expenses, it can become your full-time job.

And for the contributors, everyone, as one of the maintainers of small libraries, I truly appreciate your time spent writing PRs, making improvements, and even doing something as small as raising GitHub issues; it really means a lot to the maintainers.

And last but not least, people are using Open Source. I appreciate your trust in the Open Source project and your willingness to try and implement it in production. This gives the maintainers a sense of fulfillment. Let's celebrate this Open Source month with more contributions and the use of Open Source projects.

@camilamaia – Camila Maia

github.com/camilamaia
maintaine.rs/camilamaia

Breaking Barriers: Unlocking Opportunities Through Open Source

Have you ever found yourself stuck in the endless loop of needing experience to get a job, but struggling to get experience because no one will hire you without it? It's a typical case of a Catch-22 — a paradox where you need something you can't get without already having it. It's a frustrating cycle that many people in tech — especially those from underrepresented groups—face. You may feel like you've hit a wall, unable to break through. But what if the very thing you're missing—experience—is something you can create on your own?

I'm Camila Maia, a backend developer who's been in tech since 2010. For the past few years, I've been diving deep into Developer Experience and dedicating a good part of my time to contributing to and advocating for Open Source. I'm also Brazilian, a woman, a lesbian, and a person with a disability (low vision) — sharing this feels important, especially for those who might be looking for someone they can relate to.

In this article, I'll take you through my journey in the Open Source world — how it's shaped my path, continues to guide my professional choices, and keeps influencing what's ahead. I'll also share how it helped me find purpose in my career, something I couldn't achieve while working exclusively with private code in companies.

On top of that, I'll explore how Open Source is not just a tool for development, but a powerful vehicle for creating opportunities and empowering others — whether through teaching, mentoring, or opening doors that weren't available in more closed environments.

Open Source, Camila. Camila, Open Source

It all started during my time at Loadsmart, a logistics company where I spent nearly four years. It was there that I met Gustavo Barbosa — and everything started to shift. He introduced me to Danger, an open-source tool that automates tasks and highlights issues during code reviews, and a plugin he had created, danger-android_lint, which integrates Android Lint checks directly into pull requests. From the moment I saw it, I was hooked.

At the time, around 2018, GitHub Actions wasn't around yet. The idea of automating Pull Request reviews in such a simple, flexible way felt like magic. But what truly fascinated me was the global collaboration I saw happening. People from different corners of the world working together on the same codebase, even though they had never met — that blew my mind. It was this sense of community that drew me in and made me realize the power of Open Source.

One day, we spent hours trying to fix a bug in Danger and just couldn't crack it. While I was visiting Loadsmart's New York office, we decided to check out a CocoaPods Peer Lab Meetup at Artsy — hoping we could get help. There, we paired with Orta Therox, the creator of Danger, and together we finally solved it. Sitting with him, learning, chatting, eating pizza — it was surreal. That experience showed me firsthand how Open Source can lead to genuine connection, shared knowledge, and opportunities that might never happen in more traditional tech spaces.

As I immersed myself more and more in the Open Source world, I started hearing stories that inspired me deeply. I learned about Felix Krause and his tool Fastlane, Max Stoiber and his project react-boilerplate, and many others who had built incredible things. Each story added fuel to the fire — I knew I wanted to be part of this movement.

Born from the Fire: My Own Open Source Spark

It was mid-2019, and I was playing the "firefighter" role on my team — the person responsible for fixing bugs and keeping things from catching fire while everyone else focused on building new features. That sprint, though, the number of bugs skyrocketed. As I dove into the chaos, a pattern started to emerge: most of the issues stemmed from broken communication between services — mismatched data, outdated documentation, and APIs that just didn't behave as expected.

Debugging was painful. To test a single endpoint, I often had to recreate an entire chain of previous requests, sometimes across services we didn't even control. That's when I started thinking: there has to be a better way. I sketched out what this "better way" could look like — something to test both owned and third-party APIs, generate up-to-date documentation automatically, and even chain requests together. I spent a weekend building a proof of concept and shared it with my team on Monday.

That project became ScanAPI. I built it from the beginning with Open Source in mind, designing it to be language-agnostic and easy to extend. It started as a tool to solve our pain, but quickly grew into something bigger — a way to help anyone, anywhere, ensure their APIs actually work the way they're supposed to.

A Door Opens

By 2021, things were starting to shift. I had spoken about ScanAPI at several national and international conferences, and the response was overwhelming. People weren't just interested — they were captivated. I started joining live streams to showcase the tool in action, and every time I hit "run," I could see the excitement in the chat. The GitHub stars kept climbing. My little firefighter side project had turned into something that genuinely helped people — and they were telling me so.

In March of that year, I left my job. With more free time on my hands, I decided to fully dedicate myself to ScanAPI. I didn't know exactly what I was aiming for, but I knew I wanted to see how far I could go if I gave it my full attention.

Then, in May, something unexpected happened.

I received an email from someone named Daniel Compton at GitHub. It wasn't spam — it was real. New Relic, a major tech company, wanted to sponsor my GitHub profile. I was stunned. It was the first time someone wanted to *financially* support what I was building. It felt surreal.

There was just one problem: GitHub Sponsors wasn't available in Brazil yet. After a few back-and-forth emails, Daniel told me:

> "We're working to enable Brazil as an option for Sponsors so you can sign up. You'll be the first person in Brazil to join!"

That sentence stuck with me. *The first person in Brazil.* I couldn't believe it. Not only was I getting sponsored — I was also paving the way for other Brazilian developers to follow. It felt bigger than me.

And still in May 2021, it happened. I officially became the first Brazilian to join GitHub Sponsors. It was a spark. A glimmer of something I had never thought could be real: what if I could *live* from Open Source? What if the work I loved, the work that felt most meaningful to me, could also sustain me?

Later that year, in December, ScanAPI received a one-time donation of $1,000 USD through GitHub Sponsors. It came from Red Hat. No email. No announcement. Just support. Back in October 2020, I had given a talk about ScanAPI to their internal engineering team, invited by Og Maciel. I had no idea it had made that kind of impact. But apparently, it had.

That was the beginning of a dream. A dream to live off Open Source. Not just for me — but so that others, especially those who come from places and backgrounds like mine, could see that it's possible too.

The Cost of a Dream

Dreams can be powerful — but they also come with a price.

After the initial excitement — the talks, the stars, the sponsorships — reality started to knock. And this time, it didn't ask for permission.

The financial support I was receiving from GitHub Sponsors was meaningful, but it wasn't enough to cover my basic living expenses. I had poured everything into a project I truly believed in, but I couldn't make it sustainable — at least, not alone.

At the same time, I found myself in a strange paradox: collaborating with developers from all over the world, yet feeling deeply alone when it came to key decisions about the project. As the project grew, I became increasingly aware that I didn't want to be the only person driving it forward. I didn't *want* to be the bottleneck. I actively tried to create some kind of shared governance — spaces for discussion, roadmapping sessions, open invitations to co-lead — but none of it really stuck. Despite my efforts, I often found myself making big architectural decisions alone, shaping the roadmap in isolation, unsure if I was heading in the right direction.

People were contributing, and I'm incredibly grateful for that. But the help mostly came through peripheral issues — bug fixes, improvements to documentation, isolated features, CI updates. Those contributions mattered. They kept the project alive. But when it came to the core of ScanAPI — the deep, structural parts of the code, the long-term vision — it was still just me.

There was a gap I couldn't bridge. Developers with the experience to dive into those complex questions were often fully employed and had little spare time. Meanwhile, those who did have time to engage more deeply were usually still building up the experience needed to navigate such decisions.

And then, beyond code, there was life.

The situation in Brazil was tough. My family and I were feeling unsafe, and the political climate only added to our anxiety. We started considering a move — somewhere we could breathe easier, walk down the street without fear. But making that kind of life change takes money, planning, and stability — three things I didn't have while trying to live from Open Source.

A New Old Path

The hunt for a new job had begun. I was specifically targeting companies based in Berlin — or at least ones that would allow me to live there. I was looking for a place where I could feel safe, where inclusion and diversity weren't just buzzwords, and where Open Source wasn't an afterthought, but something genuinely valued and encouraged.

Throughout this search, I kept coming back to the impact Open Source had already had on my career. I realized early on that my contributions were immensely valuable in hiring processes. These weren't just code commits — they gave me access to real, meaningful experiences. Through Open Source, I had found myself managing people, projects, and even events. I built leadership skills, collaborated with diverse teams, and took on responsibilities that extended far beyond the technical. All of that gave me concrete, lived examples to draw on when facing tough interview questions — not hypotheticals, but stories rooted in actual work.

I came across some open roles at SoundCloud, and it immediately stood out. I already knew about their legacy — they were the birthplace of Prometheus, one of the most widely used monitoring systems in the world. That alone said a lot. But what really caught my attention was their approach to time: they had a practice called Self-Allocated Time (SAT), previously known as Hacker Time. SAT is a structured policy that gives engineers dedicated time — every week — to explore, create, learn, and contribute beyond the daily backlog. It's not just tolerated; it's part of how they work. And that includes contributing to Open Source.

I also had personal connections at SoundCloud, and through them, I knew their commitment to diversity and inclusion was genuine. It wasn't just a marketing ploy.

I applied. I got the offer. I relocated. In January 2022, I joined SoundCloud — finally in a place where Open Source wasn't just allowed, it was celebrated.

A Wild Side Quest Appears

Before we continue, let's take a quick break from the main timeline and rewind a bit — back to July 2021. My cousin — Maria — asked if we could schedule a one-on-one — she had a bunch of questions about tech. After trying many different paths in life, she was now considering a career change to something more stable, and I.T. was on her radar.

That conversation turned into a series. We started meeting regularly with a clear goal: figure out if she would actually enjoy working in tech.

Making a career shift is never easy — especially in places like Brazil, where you often can't afford to stop working to study full-time. The economy doesn't leave much room for risk. So before she flipped the table and changed her life completely, we needed to be sure this was the right path for her.

We began with the basics. I explained the different areas within tech, what day-to-day work looks like in each one. It was mostly theory, and we took a few months there. But eventually, theory wasn't enough. We needed to test it in practice.

She applied to several bootcamps and actually got accepted into a few. But most of them required a heavy time commitment, and she couldn't afford to leave her job. She eventually found one

that fit her schedule and completed it — but even then, everything still felt too abstract. She couldn't picture what working in tech actually looked like: being part of a team, collaborating on a codebase, navigating real-world tasks and communication.

That's when it clicked: **Open Source**.

Open Source could give her a hands-on experience — a real glimpse into the day-to-day life of a developer — *before* having to go all in. It would let her see the workflows, the tools, the communication style, the problem-solving... all of it.

From there, I started building a whole methodology to teach her how to code — and Open Source became the foundation of the practical phase.

Open Source: A Powerful Educational Tool

For Maria's hands-on learning journey, we chose to work on brutils-python — a utilities library focused on Brazil-specific data formats. With it, you can validate, format, and generate common national identifiers like personal tax IDs (similar to a Social Security Number), business registration numbers, vehicle license plates, voter IDs, and more.

It was the perfect project: no complex frameworks, no web development, and highly modular features. Ideal for practicing core software development skills in a real-world environment.

In February 2023, Maria made her very first contribution.

Step by step, she learned how the library worked — how to contribute to it, how to maintain it, how to think collaboratively and build for the long term. By October of that same year, during Hacktoberfest, she was already coordinating a group of over ten people contributing simultaneously.

We started sharing the project and this journey in conferences and tech communities. Over time, the impact grew: the library reached over 4,000 downloads per month.

Maria's confidence as a developer soared. She fell in love with Open Source — and was honestly amazed that people would just show up and contribute to a project they found on GitHub. It gave her a real sense of working in a team, building something that mattered.

Then came something unexpected: she began receiving messages from recruiters, referencing her GitHub profile and her work on brutils. For someone who had never worked in tech before, it was incredible.

With all of this unfolding, we couldn't help but ask: *what if this same approach could work for more people?*

Cumbuca Dev

In August 2023, after everything we had learned from Maria's journey, we decided to take things a step further — and that's when Cumbuca Dev was born.

Cumbuca Dev is a community-driven educational initiative focused on helping people from underrepresented groups gain real experience in tech, especially through Open Source. We believe learning doesn't have to happen in isolation, and that meaningful experience doesn't have to wait until after you land your first job.

One of our core goals is to break a cycle that keeps so many people out of the industry: **you need experience to get a job, but you need a job to get experience.**

With Cumbuca Dev, we flip that script. Through guided contributions to real projects, mentorship, and collaboration, we help people build skills, confidence, and visibility — even before they write their first résumé.

It started with one person. Now, it's growing into something bigger.

Old Dream, New Strategy

I spent almost two years working at SoundCloud. It was a time of deep learning and meaningful connections — I met incredible people and, for the first time, got to work in a truly diverse team. But even with all the positives, I couldn't contribute to Open Source as much as I wanted to through the company.

In December 2023, I joined Trade Republic. But it didn't take long for me to realize: my path was no longer about working for a company. Cumbuca was taking shape. New projects were blooming. And my time simply wasn't enough anymore.

So, I made a decision: In May 2024, I left Trade Republic to fully dedicate myself to Cumbuca — and I didn't stop there. I also moved back to Brazil. Cumbuca was born with Brazil in mind — to help break cycles of exclusion in a country full of potential but short on opportunity. Being closer to the people we aim to support, especially during these crucial first years, was vital. Building real connections, understanding the local context, and being physically present in these spaces matters, especially when your goal is to create something truly grounded in the local reality.

The dream remained the same: to live from Open Source. But now, it had expanded, and so had its purpose. It wasn't just about me anymore; it was about teaching, supporting, and opening doors for others.

We don't want to repeat the mistakes of the past — creating Open Source projects that depend on a single person to sustain them. That model is unsustainable, burns people out, and limits

the collective impact we can have. This time, we're focused on building a community — one that supports itself and grows from within. We want to foster Open Source in Brazil — and, naturally, extend that impact globally. We believe that if someone's first experience in tech is through Open Source, they'll feel its impact from day one. And when that happens, they're far more likely to keep contributing — becoming part of the cycle and helping grow the community from the inside out.

At the same time, we understand that tech isn't for everyone — and that's okay. One of our goals is to help people realize this as early as possible. If we can save someone time, energy, and self-doubt by helping them discover that early on, then that's a success too.

We're not doing this alone or in silence. We offer mentorships to support people on their learning journey, and we seize every opportunity to spread the word about Open Source — in conferences, local events, meetups, and communities. And we're beginning to explore the value of Open Source inside companies — showing its potential and broad impact across teams and the tech ecosystem.

Building from the Basics

Now, we're building the kind of ecosystem we wish we had when we started.

We're creating free, accessible content in Portuguese to teach people not only how to code, but also how to be part of the Open Source community — from the very beginning. From the basics. No assumptions, no prior knowledge required. Just clear, welcoming, beginner-friendly guidance for folks who are just taking their first steps in tech.

And we're not stopping at content.

We're also building and maintaining repositories with real-world scale and complexity — offering practical, hands-on opportunities to learn by contributing to something that matters.

While we're starting in Portuguese, we plan to gradually translate everything into English too. It takes time, but we believe it will pay off — and that by building strong foundations in our own language, we'll create more powerful contributions to the global community.

One of the first steps in this journey is Git e GitHub para Humanos — our first public book, written entirely in Portuguese.

The name roughly translates to "Git and GitHub for Humans", and that's exactly the point: making these tools feel approachable to people who are just getting started — especially those who are still learning to program and might feel overwhelmed by all the jargon out there.

The book is fully focused on helping people contribute to Open Source. It brings practical guidance on how to use Git and GitHub in real Open Source workflows — including tips on

good practices, how to structure contributions, understanding how repositories work, and how to communicate effectively when opening issues or pull requests.

What makes this guide different is the tone: human, welcoming, and clear. No gatekeeping, no overly complex explanations — just practical knowledge, simple language, and examples that actually make sense in real life.

The guide is still in development, but it already marks the beginning of something much bigger. We know that, for now, this means taking one step back — spending less time maintaining the code itself, and more time building the foundations around it. But we believe this is the only way forward: to grow a community that doesn't just use Open Source, but understands it, contributes to it, and helps it scale — together. It's the first building block of a broader ecosystem we're creating — one that lowers barriers, invites more people in, and helps foster a stronger, more inclusive Open Source culture in Brazil and beyond.

The Ongoing Journey

As I reflect on this journey, I see how Open Source has transformed not just my career, but my personal life as well. It's been a vehicle for growth, connection, and empowerment, not just for me, but for those I've had the privilege to mentor and collaborate with. My journey continues, and I'm excited about where it will take me — and where it can take others. I've seen firsthand how Open Source can break barriers, provide opportunities, and allow people from all walks of life to shine. My cousin Maria's transformation from a beginner to a leader in the Open Source community is just one example of the profound impact it can have.

If there's one thing I've learned through all of this, it's that Open Source is more than just code. It's about people. It's about building a community where everyone, regardless of background or experience, has a chance to contribute, grow, and make a difference. And that's why I'll continue to invest in it — not just for my own growth, but to ensure that the doors Open Source has opened for me remain wide open for others too.

I want to take a moment to thank everyone who has contributed to this journey — whether by sharing knowledge, offering guidance, sharing your story, or simply being part of the community. Your support has been invaluable, and I'm incredibly grateful for each and every person who has helped shape this experience. It's through collaboration, mutual support, and the inspiring stories we share that we motivate others and continue to move forward together.

If you're curious to learn more about the projects we're building, feel free to explore cumbuca.dev, our GitHub organization at github.com/cumbucadev, and some of the initiatives close to our hearts like ScanAPI, brutils-python, and Git e GitHub para Humanos. We'd love for you to join

us on this journey — whether by contributing, learning, or simply sharing the passion for making Open Source more inclusive for everyone.

Open Source is a powerful way to break the First Job Catch-22. If you feel stuck, know that you're not alone — the community is here to help you create your own opportunities. Just start contributing, and you'll be amazed at how doors begin to open.

Camila Maia

@cezaraugusto – Cezar Augusto

github.com/cezaraugusto
maintaine.rs/cezaraugusto

I have wanted to work with Open Source since I first heard the term. I've always had a philosophical view of it — the idea of software being shared with the intent of being useful to others, the ability to contribute to projects from all over the world, and knowing that something you built can positively impact people you may never meet in person.

In my career I always wanted to be useful and produce useful work. Open Source makes me feel that way.

I'm Cezar Augusto, and I created Extension.js, an open-source tool that makes it very easy to create cross-browser extensions.

Open-source journey

I started contributing to Open Source by triaging bugs and writing or translating documentation — mostly for projects related to front-end development and browsers, which have always been my core interest. I've always been fascinated by how browsers work, and that obsession led me to contribute to projects like MDN and the iconic Front-End Developer Interview Questions, which I later helped maintain.

At the time, I wasn't very confident in my code. But once I gained that confidence, I started looking around ways to contribute code to an open-source browser. These were the early days of the Brave Desktop Browser, which gave me my first meaningful contribution and my first real contact with browser extension development.

Extension.js

Browser extensions have existed for a while, but surprisingly few people know how they work or how to build one. Once I got familiar with the ecosystem and started teaching colleagues and friends, I realized that depending on the browser and the context where your extension runs, things we take for granted in web development — like live-reloading or importing static files — aren't easily supported.

Back then, you could rely on boilerplate projects to get started, but they required too much configuration. If you wanted to use a modern stack like React, you'd have to learn tooling that felt disconnected from most web workflows. That made learning browser extension development a frustrating experience.

Developing browser extensions shouldn't feel like stepping back a decade in tooling. Extension.js changes that. It brings modern, zero-config tooling to the world of browser extensions — so anyone using TypeScript, WASM, or frameworks like React, Svelte, or Vue can build once and ship to all major browsers without compromising the developer experience.

Used by hundreds of projects, Extension.js is a tool I'm proud to see helping other developers build browser extensions with ease.

Challenges

Extension.js began as a solution to a personal need, and it's continued to grow alongside my career. But it needs to evolve. Maintaining an open-source project taught me something no book or job ever could: progress often happens quietly. The most important work is often invisible — reviewing issues, refactoring silently, or managing expectations. It's not glamorous, but it's what keeps projects alive.

As a maintainer, I'm constantly balancing shipping with listening, and personal growth with community needs. I'm often eager to ship a feature or fix a bug, but the project is now bigger than me — and that means being more thoughtful about the impact of every decision. It's not easy, but it's deeply rewarding.

AI

Artificial Intelligence empowers developers in ways that make projects more accessible to build, allowing individual maintainers to ship features, debug issues, and write documentation at a pace that once required a full team.

It also makes contributions from both seasoned engineers and newcomers equally impactful, which is a huge help for solo maintainers or small teams working on larger open-source projects.

The next generation of open-source projects won't be built by teams — they'll be orchestrated by individuals augmented by AI, opening doors to diverse contributors and lowering the barrier to innovation in ways we've never seen before.

Advice

You don't need permission to make a difference. Open Source rewards curiosity, generosity, and consistency. Whether you're a first-time contributor or a seasoned maintainer, your work matters — and it's often the quiet commits that make the loudest impact.

For developers looking to contribute: if you want to work with Open Source because you believe you can do a good job in service of something greater than yourself, do it. Don't expect recognition or compensation. The biggest joy I get from working in Open Source is knowing the impact my code can have.

For contributors and maintainers: make your project easy to contribute to, and take the time to understand the people who use it — they are your biggest contributors. Your code is open to everyone, and there's always a chance someone might copy or rebrand it. That could be an unoriginal developer or a large corporation. Assume this is a sign of great software — and potentially the best compliment your work can get.

For everybody else: acknowledging good work is one of the simplest, most meaningful actions you can take to honor someone's effort. Whenever you can, contribute — whether with code, knowledge, promotion, or anything you're good at — to your favorite projects and developers.

To all my fellow maintainers and open-source contributors: thank you — your quiet dedication powers the tools, ideas, and communities that move the web forward.

@darccio – Dario Castañé

github.com/darccio
maintaine.rs/darccio

Hola! I'm Dario Castañé, a software engineer and lifelong Open Source enthusiast based in Catalonia. In my career I've worn many hats – from full-stack developer to engineering manager – but a constant through it all has been my love for Free/Libre Open Source Software (FLOSS). I'm now a Senior Software Engineer at Datadog, where I work on Open Source client libraries (specifically the Go APM tracer). My journey into Open Source began with a simple desire to share solutions to problems I encountered.

Some of my projects

Over the years, I've created and maintained several Open Source projects that reflect my diverse interests:

- **Mergo** – a tiny Go library for merging structs and maps. I released Mergo back in 2013 to help configure default values in Go applications, and it took on a life of its own. Incredibly, Mergo is now used by over 60,000 repositories on GitHub and has been adopted by major projects like Docker, running my code millions of times in production.
- **Asembleo** – a pseudo-anonymous voting system for general assemblies and organizations. I built Asembleo inspired by my interest in civic tech and grassroots democracy. As a former city councilor in my hometown, I wanted a secure Open Source tool to help communities make collective decisions. Asembleo combines my political passion with coding, enabling transparent votes in a way that protects privacy. It's an example of how Open Source can strengthen democratic participation.
- **Zas** – the simplest static site generator you can imagine, written in Go. Zas powers my personal website and blog. Rather than use a big framework, I created Zas to generate my site with just the features I needed. It's minimalistic by design – the joy was in building something from scratch and sharing it. Zas embodies the "indie hacker" spirit I love: if the tool you want doesn't exist, why not create it and Open Source it for others?

These projects (and a few others in my GitHub) each started as a personal itch to scratch. By releasing them openly, I discovered the joy of seeing others benefit from my code. Each

repository became a little community of its own, where I could collaborate with users and other contributors.

Reflections on impact and community Contributions

When I look back, I'm humbled by the impact some of my work has had. For instance, I never imagined that a small utility like Mergo would become part of fundamental systems. Knowing that my code runs in data centers worldwide is both exciting and a little daunting. It really underscored for me how **a single Open Source contribution can ripple out to millions of users**. This realization has been one of the most motivating aspects of being a maintainer.

Beyond code, I've tried to give back to the community in other ways. I'm an active advocate for Open Source, open access, and free culture, frequently speaking at meetups and conferences about the importance of sharing knowledge. I've organized local tech meetups, and I also volunteered in grassroots initiatives campaigning for fair legislation around copyright laws and Open Source. For me, Open Source is not just about code – it's about a set of values: **collaboration, transparency, and empowerment** of individuals and communities.

Maintaining Datadog's dd-trace-go

In my professional role at Datadog, I'm part of the team maintaining **dd-trace-go**, Datadog's Go client library for APM (tracing, profiling, etc.). This has been a unique experience because it sits at the intersection of corporate software and Open Source. On one hand, dd-trace-go is critical to many companies' infrastructure (including our own product), and on the other hand it's Open Source on GitHub, with a community of users and contributors just like any other OSS project.

Maintaining dd-trace-go has reinforced a few key lessons for me:

- **Consistency and reliability are paramount:** When thousands of businesses rely on your library for monitoring their systems, you can't afford to break things. We are extremely careful with backward compatibility and thoroughly test every change. I apply the same rigor in dd-trace-go that I do in my personal projects: writing extensive tests and using linters to catch issues early. This discipline was something I cultivated through Open Source, and it pays off hugely at enterprise scale.
- **Community feedback is gold:** Being an Open Source maintainer at a company means we get constant feedback from external users – bug reports, feature requests, even occasional pull requests from the community. I've learned to embrace this feedback loop. Users of dd-trace-go often surface use-cases we hadn't thought of. By listening and engaging with

them, we improve the library in ways that benefit everyone. It's a virtuous cycle: an Open Source ethos inside a commercial product.

- **Collaboration and mentorship:** Within our team, we treat dd-trace-go as a shared responsibility. We review each other's code, collaborate on design decisions, and mentor newer engineers in Go best practices. I've found that my experience in Open Source – where code review and knowledge sharing are the norm – prepared me well for this. A healthy maintainer team functions much like an Open Source community, where respect and continuous learning are key.

The importance of Open Source supply chain management

One aspect of Open Source that has really hit home for me is the importance of the **software supply chain** – the network of dependencies and libraries that modern applications rely on. As maintainers, we are not just writing code for ourselves; we're effectively stewards of a supply chain that others trust. I learned this dramatically through Mergo. At one point, because of one teeny-tiny mistake in an update, I inadvertently *broke a released version of Docker*. It was an "oops" moment that taught me how even a minor change in a widely used library can have far-reaching consequences, even when you have an extensive test suite in place.

That incident turned into a story I now share with fellow developers: always consider the downstream impact of your changes. I was fortunate – the Docker community and maintainers were understanding, and we worked together to fix the issue quickly. But it highlighted the **responsibility maintainers carry**. Since then, I pay extra attention to semantic versioning, changelogs, and testing against real-world scenarios. It's crucial to communicate breaking changes clearly (or avoid them when possible). In Open Source supply chains, trust is everything – users trust that our component will function as expected and not compromise their systems.

Security is another big part of supply chain management. I've become much more proactive about addressing security reports and keeping dependencies up to date. In the wake of high-profile supply-chain attacks and vulnerabilities in recent years, I feel it's part of my duty as a maintainer to ensure my projects don't become weak links. This means embracing tools and best practices for dependency management, auditing, and incident response, like the ones championed by OpenSSF. It's not the most glamorous part of Open Source, but it's absolutely vital now.

Ultimately, my experiences – from the Docker mishap to managing dd-trace-go – have reinforced how interconnected the Open Source ecosystem is. We're all links in a chain. By strengthening our own projects, we help secure and stabilize the broader ecosystem.

Closing thoughts

Looking back at my journey, I feel incredibly grateful for the Open Source community. Open Source transformed my career and even my outlook on life. It taught me that sharing knowledge openly can create enormous value – often in ways we can't predict. As I once phrased it, **releasing code into the wild is an act of kindness**, a way to help others scratch the same itch you had. You may never know how or by whom your work will be used, but that's the beauty of it. Your small project might become a building block in someone else's dream.

Open Source is a two-way street: you give something, and you almost always get something unexpected in return, be it new knowledge, friendship, or the satisfaction of solving a tough problem. In my journey from hacking on side projects to maintaining major libraries, that lesson has been constant. **Contributing and sharing in public is worth it** – for you, for others, and for the sheer progress of technology.

@delta456 – Swastik Baranwal

github.com/delta456
maintaine.rs/delta456

I'm **Swastik Baranwal**, an Open Source Developer. I've contributed to several Open Source projects and collaborate with maintainers across the ecosystem.

My Journey

I was introduced to Open Source in 2019 through Hacktoberfest. That's when I discovered The V Programming Language and saw people building an entirely new language. I was fascinated by their dedication.

Just 12 days in, I made my first PR by implementing some basic string methods. I was amazed by how committed the community was. I began contributing and collaborating with them while learning to use tools like `git`, `make`, GitHub, and other essential development utilities.

Now, *five years* later, I maintain several projects and help newcomers get into Open Source.

Projects I'm Involved In

I've been part of The V Programming Language since the beginning of my Open Source journey and serve as one of its main developers. I've implemented features like operator overloading, syntax highlighting, support for match branch expressions, and fixed several compiler issues. I remain highly involved in the community.

I also maintain a personal project called Box CLI Maker, which helps draw and use boxes in terminal applications. It's used in several packages, notably in **Kubernetes's minikube**. I even appeared on a podcast discussing it during GitHub Open Source Friday. I plan to continue improving and expanding the project.

I actively contribute to the WebDriver ecosystem—including Selenium, Appium, and WebDriverIO—as part of my work with LambdaTest OSPO. I've implemented many features and fixes, especially in the *Python* and *Java* bindings of Selenium, and work closely with other maintainers.

Beyond that, I'm involved with projects like the TODO Group, charm, nixpkgs, catppuccin, and others in areas such as Open Source Governance, Terminal UI, CLIs, WebDriver, Compilers, Low-Level Programming, and DevTools.

Challenges I've Faced as a Maintainer

One major challenge has been reviewing *massive* PRs with critical changes. These reviews take a lot of time, and I must ensure they don't negatively affect the project.

Another pain point has been dealing with CI delays or failures, even when everything works locally. That wastes time and disrupts my workflow.

Onboarding new developers is particularly tough. It requires a lot of time explaining the workflow, helping them navigate the codebase, and often working alongside them.

And of course, limited or no sponsorship can impact maintainers significantly. It can force people to abandon their own projects to remain financially sustainable.

Ultimately, Open Source needs **more maintainers** to pass on knowledge and help projects thrive independently of their original creators. That's how we ensure sustainability.

How Contributors Can Support Maintainers

In my view, contributors can help in both code and non-code ways.

Coders can add features, fix bugs, and improve the project technically.

Non-code contributors can support documentation, keep guidelines and websites up to date, enhance design, and help manage changelogs.

Financial support also matters. Sponsorships give maintainers more time to work on their projects.

While this varies with the scale of the project, even small non-code contributions can make a big difference.

The Impact of AI on Open Source

AI's ability to write code has greatly influenced Open Source—both positively and negatively.

On the plus side, contributors can work faster and projects evolve more quickly. Tasks get completed faster, and onboarding becomes less time-consuming.

However, AI has also increased the volume of spammy issues and PRs. Some people want to contribute but skip learning the project's workings, relying on AI-generated code that often misses the mark. Reviewing irrelevant PRs wastes time and energy.

AI is still new, and we maintainers are figuring out how best to harness it—while also managing the downsides.

Growing Communities

While I mainly focus on developing Open Source projects, I also support local communities and help them get into Open Source.

I volunteer with **FOSS United**, a non-profit that promotes and strengthens the Free and Open Source Software (FOSS) ecosystem in India.

FOSS United organizes meetups, city conferences, an annual conference, college events, and hackathons. It also offers grants to Indian Open Source projects.

I help organize meetups for FOSS United Delhi, a city chapter. I manage venue arrangements, speaker outreach, and community engagement for these events.

I believe FOSS United will become a foundational pillar of India's Open Source ecosystem. I'm proud to be part of it and honored to have been elected to its *Governance Board*.

Connect with Me

Want to see what I'm working on next?

- **Twitter**: https://twitter.com/Delta2315
- **GitHub**: https://github.com/delta456
- **LinkedIn**: https://www.linkedin.com/in/swastik-baranwal/

If you like my open-source work, then feel free to sponsor me via GitHub Sponsors

@derberg – Lukasz Gornicki

github.com/derberg
maintaine.rs/derberg

My first active interaction with open-source happened back in 2014, and I was immediately captivated by the concept. The idea of people collaborating openly, across borders, with a shared goal of creating something better fascinated me. At the time, I didn't contribute much code, largely due to imposter syndrome, which was in full swing. Yet, even then, I sensed that contributing to open-source wasn't just about writing code. People can give back in so many other ways.

So, I found my own path, helping by securing funding for one of the projects and promoting tools at conferences.

Everyone needs a role model—someone to inspire or spark that first sense of purpose. For me, that person was Benjamin Lupton. We were using DocPad, and after watching Ben's DocPad presentation, I knew I was slowly becoming an open-source enthusiast, or maybe even a fanatic.

What's Open Source to you?

A diverse community that fosters innovation. As simple as that.

There is no other place, no company that can give you this. The ability to work together with people from different cultures, different sides of the world. The ability to work with people having different experience, different points of view creates an environment where innovation happens.

What projects are you involved in?

I'm active in the AsyncAPI Initiative, and my highest priority is the AsyncAPI spec and the AsyncAPI Generator. At the moment, I'm the Executive Director of the initiative, but I'm stepping down in favour of our new Governance Board.

How do you grow your community?

This is a long story to tell. In short, by putting the community first and everything else later, the most successful programs we've implemented to grow the community are:

- **Dedicated participation in mentorships** (GitHub). We also set up our own mentorship program. We're pushing for a new concept we call `Maintainership`, which focuses not on creating new projects but on mentoring people on how to become a maintainer and how complex and responsible a role it is.
- **AsyncAPI Conference** (conference.asyncapi.com) – rather than a single annual event that requires extensive travel, we hold multiple events and host smaller sessions at well-established conferences. This approach allows us to reach a larger, more diverse audience in different locations. I recommend you read in detail how from an online one-day event we scaled to most of the continents with multiple events a year so you can learn how such an approach can help you gain funding for the projects and friends that help to scale your community.
- Build a program that recognizes people who promote your project. Call it Ambassador, Champion, Hero, Jedi - whatever you like, just recognize these people and give them space. You will never be able to promote the project alone as much as in a group of various influencers. Our Ambassador program gathers people who talk about AsyncAPI at conferences, record videos, write articles - basically create publicly available content.

We also run other community programs.

What are the main challenges you face as a maintainer

Financial sustainability. You can sponsor me of course or hire my services to fix that.

What are some ways contributors can better support maintainers

- Read the contributor guidelines :)
- Be proactive: research first and ask thoughtful questions.
- Remember: your new feature might be great and solve your use case, but ultimately I will be the one to maintain it. Writing code is easy; maintaining it is much harder. Keep this in mind when you urge maintainers to merge something and avoid putting yourself in a privileged position, expecting only gratitude instead of healthy skepticism.

Honestly, if every contributor at least followed the first bullet point, the world would be a better place! :)

What are some of the key security practices you've implemented in your project

I highly rely on external services that are free for Open Source and verify overall security of the project and check changes per pull request.

I'm mainly focusing on making sure our secrets do not leak, and that we use GitHub Actions in a secure way.

Much more could be done though.

What do you think are the biggest security challenges facing Open Source today

The biggest challenge is that users of open-source software expect maintainers, who aren't paid for their work, to be fully responsible for producing secure software.

Personally, I'm also concerned about the threat posed by potentially malicious maintainers.

What's the impact of AI on Open Source development?

Too many contributors use it incorrectly and end up spamming projects. But for me, as an experienced maintainer who can verify AI-generated output, it speeds up my work.

Does this speed-up balance out the AI spam? Does it mean nothing has changed? I have no idea. :)

@desrosj – Jonathan Desrosiers

github.com/desrosj
maintaine.rs/desrosj

Hello, my name is Jonathan Desrosiers. I've been a credited contributor to the WordPress project since 2013, a Core Committer since 2018, and a maintainer of several components throughout that time.

I first encountered Open Source in college, using WordPress to build some websites for myself, school projects, and some freelance clients. That experience eventually led to a day job building WordPress sites, and soon after, I began attending and speaking at WordCamps.

One day, I found a bug that was affecting my work. Instead of working around it, I submitted a patch. That sparked a deeper interest in how the software was built and maintained. I've been contributing to Open Source ever since.

Because WordPress overlaps with many other Open Source projects, it's common to discover upstream problems. This frequently leads to submitting bug reports or patches to other code bases. Even though each project has its own goals, there's a shared sense of collaboration and stewardship across the ecosystem that's both gratifying and contagious.

This essay is based on a talk I gave at WordCamp Europe 2025 titled *"How a Core Committer Thinks: Making Decisions for Millions,"* which was heavily influenced by the works of Havoc Pennington and Karl Fogel. It also includes some thoughts from a blog post I published reflecting on the keynote session at the same event.

Software changes lives. Often in unanticipated ways.

This is especially true with Open Source software. Open Source can be a creative outlet. It can empower you to transform your career, access jobs you previously couldn't, find mentorship, feel a sense of belonging, start a business, or help others do the same.

Open Source is about coming together despite our differences to accomplish a shared goal and publishing it for the benefit of the world. When we tackle problems together, the solutions we build are more resilient, more innovative, and more impactful than anything we could create in silos.

Maintaining Open Source software and making decisions that affect every user in unique ways is both a burden and a privilege. Just the thought of this scale can sometimes make committing code terrifying, even to seasoned maintainers. But having decision-making frameworks and foundational philosophies in place help ensure that we make the best choices we can for our users.

We should all strive to understand how the software we rely on in our personal and professional lives approach change and manage risk while making decisions. Here's how the WordPress core philosophies are used to guide maintainers when making decisions.

Maintainers in the WordPress Project

As of June 2025, WordPress powers 43.5% of *all* websites and 61% of websites using a known CMS.[1] Businesses, doctors, banks, governments, individuals, and nonprofits are just a few of the many stakeholders that rely on the project's maintainers to deliver stable, reliable, and effective software across countless use cases.

In the WordPress project, a Core Committer is a trusted contributor that has been granted write access to the canonical WordPress code base. In addition to reviewing and authoring changes to the code base, they are also responsible for upholding the project's philosophies, mentoring contributors, keeping the project on track, and deeply considering the impact of even the smallest change.

In the 22-year history of the project:

- 110 people have committed at least once
- 89 have committed in the last 10 years
- 55 have committed in the last 2 years
- 23 have maintained a once per month commit average over that same two-year period

There is also a second type of maintainer called a component maintainer. Component maintainers do not always have write access, but good ones exhibit many of the same qualities as committers while focusing on their small chunk of the software. Because of the overlapping responsibilities, it's common for committers to also serve as component maintainers, and for component maintainers to eventually be granted committer status.

The code base is currently divided into 43 components and 20 sub-components with 65 unique contributors actively maintaining them. Of those contributors, 37 (approximately 57%) have been granted commit access. Every Open Source community should strive to achieve a healthy balance of new, intermediate, expert, and even emeritus contributors to ensure long-term stability. But that's a topic to dive into another time!

The Pathways of Change

There are many unique ways a change can find its way into the WordPress code that is shipped to the world. Like most software, change usually takes the form of a bug report, feature request, enhancement, or task. But while a ticket is the most common starting point, not all ideas originate there. Some begin with a "what if" on a personal blog, an issue in the user support forums, or even a working group at a Contributor Day event. Let's explore three common pathways a change can move through the project before diving into the principles maintainers use to make decisions.

Tickets in Trac

Most ideas start as a ticket in the project's bug tracking software, Trac. Though antiquated in some ways, I'm fond of Trac because you must first outline and describe a specific problem in order to create a ticket. This step is sometimes skipped when solving problems in software (both intentionally and not), and can result in bad decisions or unforeseen consequences.

After the ticket is created, discussion happens in the comments or on the WordPress Slack instance. Once contributors feel that they have enough information, patches are created and attached to the ticket or submitted as pull requests to the wordpress-develop repository on GitHub. After a consensus is reached on a solution and adequate testing has been performed, a Core Committer gives a final review before committing (or rejecting) the proposed change.

Canonical Feature Plugins

While tickets on Trac are the most prevalent path, some are built out by the community in the form of a plugin before a proposal to merge the functionality into the code base is published.

A great example of this practice today can be found with the Performance Team. They maintain several feature plugins that implement new and emerging ways of improving the performance of WordPress websites. While the desired end goal is to one day include these features in the software, they can also easily continue as canonical plugins should they not be a good fit at any given time.

In the most recent major release (6.8 "Cecil"), one such feature plugin was included after over 7 months of iterating, testing, and feedback: support for the new Speculation Rules API. Once the contributors focusing on this feature plugin were happy with the implementation, a Trac ticket was opened to further discuss the problem being addressed and review the code before a committer finally authored the changeset.

The Block Editor

The block editor (also known as the Gutenberg project) uses yet another unique workflow. It is maintained as a long-running feature plugin where new functionality is added and refined. Because the block editor is primarily built with JavaScript (with some TypeScript sprinkled in), the related code is published to over 100 different npm packages. This happens every two weeks when a new version of the plugin is released to the 300,000+ sites that currently have it activated. Before each major release of WordPress, changes are merged into the canonical code base by updating the dependency manifest.

Evaluating Ideas

Having predictable workflows and expectations can be very helpful, but there is never a one-size-fits-all process. Good ideas can originate from anywhere at any time with no minimum level of experience required. As maintainers, we must always keep a sharp eye out. Even if our communities are discoverable, transparent, and approachable, ideas do not always land on our doorstep. Be willing to meet them where they are.

While process requirements can be a bit fluid, the decision-making frameworks should be more rigid. These frameworks should always focus on judging ideas based on their merit, never the identities of those who propose them. We must seek as many viewpoints as possible before making decisions.

Change is Community Driven

While only committers can merge code, they are oftentimes just a final set of eyes in a longer process. Feedback loops with users, developers, and plugin and theme authors are essential. These feedback loops when combined with direction from leadership, additional testing, documentation, iteration, and some external influence (new industry standards, versions of PHP, MySQL, etc.) drive the majority of changes in the WordPress software.

The Value of Presence

There's a premise in Open Source that underscores the value and importance of active engagement. Decisions are made by those who show up.

By participating in discussions, contributing code, submitting bug reports or feature requests, or testing proposed changes, any individual can influence the direction of an Open Source project. By showing up, you ensure that your voice will be heard. But be aware, with presence comes

responsibility. Showing up means being prepared, doing research, actively listening, and being thoughtful in your communication.

In my experience, this premise largely holds true, but there are some practical limits to this. For example, your presence grants you a voice, not necessarily a vote. As opposed to decisions being strictly made by those who show up (a do-ocracy), commit access in WordPress is meritocratic, and granted only after demonstrating a consistent track record of valuable and high-quality contributions, building long-term trust through engagement, and earning the respect of your peers.

Equal Participation

The health of a project improves when decisions are inclusive and transparent. The quality of the outcome is higher when more unique voices are heard. But we must always remember that not everyone can "show up" equally (if at all). When it comes to participating, time zones, language and cultural barriers, personal and family responsibilities, disabilities, and financial circumstances can all affect a person's ability to share their perspective. There should always be multiple ways to "show up" with reasonable time frames.

No one should be marginalized by a lack of opportunity.

The Role of Consensus

One of the most important duties of a Core committer is collecting feedback to determine the best solution for the largest number of people. No matter how good someone is at consensus building, it will almost never be perfect. Perfect is so rare that you should be suspicious when it occurs. Consider whether certain perspectives are missing or if the right questions have been asked.

> "Consensus merely means an agreement that everyone is willing to live with" [2].

In his writings, Karl Fogel explains that consensus can be either explicit or implicit. When seeking explicit consensus, always be clear what is being proposed. When someone objects, continue the discussion until the time is right to propose a new consensus. An example of implicit consensus is when a committer finds and fixes a small bug on their own. The act of committing is assuming consensus. If anyone objects, then a discussion can be had to reach a new consensus. If one can't be reached, version control is a wonderful tool that allows for easily reverting a change.

Disagree and Commit

When discussing changes in Open Source, disagreement is healthy and expected. It shows that contributors are engaged and care about the software. But endlessly rehashing the same discussions is tiresome and frustrating, and often leads to burnout.

One of the most important qualities in Open Source maintainers is the ability to disagree and commit. Even when someone disagrees with a decision, they should be able to clearly state their reasoning before publicly supporting the consensus to move the project forward over their own personal preferences.

Once a decision is made, moving forward together is essential.

Quality Logic

Hopefully, anyone can show up and create a patch for your Open Source project. If they can't, there's work to be done to improve the contributor experience. That aside, creating patches is the easy part, even when it changes thousands of lines of code.

Producing strong rationale for a change is much harder. It requires a complete understanding of what the root problem actually is, an exploration of alternative solutions in depth, and recognition of motivations. If you propose a solution before the problem is fully understood, you're doing everyone a disservice.

The best ideas are rooted in real user problems, well-scoped and practical, maintainable and testable, and compatible with the project's philosophies. It's backed with evidence, context, and potential impact while avoiding speculation. Rationale should always go beyond personal desire and novelty, and demonstrate how the change will benefit the majority of your users.

> The guiding principle is simple: ask "why," rather than "why not" [3].

A Case Study: XML Sitemaps

In WordPress 5.5 "Eckstine," a new API was added for generating an XML sitemap for every site. Let's go through the process of evaluating the rationale presented when a proposal was made to include the feature.

- Sitemaps use a consistent, de facto standard supported by all major search engines. This speaks to the maintainability and predictability of the feature. There is a widely adapted standard shaping the expectations and requirements while limiting the scope.

- 4 out of the top 15 plugins on the WordPress.org plugin repository at the time shipped their own implementation of an XML sitemap. This demonstrated a widespread demand for the feature.
- Every site should have equal opportunity to be crawled by search engines and discovered by users. This strongly aligns with the project's mission to democratize publishing.

In addition to evaluating the idea, the implementation details should also be scrutinized.

- The implementation used sane yet comprehensive defaults. All publicly visible registered post types (e.g. posts, pages) and taxonomies (e.g. categories, tags), the site's home page, etc.
- A reference to the sitemap file is automatically included in the `robots.txt` file.
- No new user facing interfaces were introduced.
- Site owners can customize their sitemap to their liking through the use of plugins or custom code.
- Sitemaps are enabled by default for all sites.

Let's evaluate this feature by applying the project's foundational philosophies.

Out of the box

"Great software should work with little configuration and setup. WordPress is designed to get you up and running and fully functional in no longer than five minutes." In this case, the feature will "just work" without any action required by the user.

Design for the majority

"Many end users of WordPress are non-technically minded." The majority of people using the software don't know or care what the XML schema for the Sitemap protocol is. These are the users we design the software for. They are the ones spending the most time using it. Applied here, all technical aspects of the feature are just handled on behalf of the user.

Decisions, not options

"Every time you give a user an option, you are asking them to make a decision. When a user doesn't care or understand the option this ultimately leads to frustration... Ultimately these choices end up being technical ones, choices that the average end user has no interest in. It's our duty as developers to make smart design decisions and avoid putting the weight of technical choices on our end users."

The Sitemap feature introduced no new options or user controls. The only way to alter the behavior of the feature is to change a pre-existing setting in the dashboard. This setting presents the site owner with one decision: should this site be visible to search engines? The code will take appropriate action to enable or disable Sitemaps behind the scenes based on this decision.

Clean, lean, and mean/Striving for simplicity

"The core of WordPress will always provide a solid array of basic features. It's designed to be lean and fast and will always stay that way... If the next version of WordPress comes with a feature that the majority of users immediately want to turn off, or think they'll never use, then we've blown it." In the project, this is also referred to as the 80% principle.

The implementation included a lean yet extensible foundation allowing plugins to easily adjust what the Sitemap includes. Despite this, "we're never done with simplicity."

The vocal minority

"The number of people who create content on the internet represents approximately 1% (or less) of the people actually viewing that content." In internet culture, this is known as the 1% rule. While it's "really important to listen and respond to those who post feedback and voice their opinions on forums, they only represent a tiny fraction of our end users."

We always need to consider and respect the massive and mostly silent user base. The fact that 4 of the top 15 plugins were shipping a Sitemap implementation demonstrated the vote of the silent majority while also clearly confirming that the feature met the 80% principle.

As a community, we should contemplate how to better engage with all users who are not yet vocal. After all, "each interaction with a user is an opportunity to get a new participant" [4].

Democratize Publishing

Supporting democratic publishing for all means not limiting the reach of someone's voice because they:

- Don't understand what a Sitemap is.
- What to include in one.
- How to best accomplish this technically.
- Don't have the means to hire someone who does.

Enabling the feature by default for all WordPress sites strongly aligns with the mission to democratize publishing.

Additional Considerations

The case study above shows how the feature strongly aligned with 6 of the 8 project philosophies. But what about the other 2? And what else should be considered when making decisions about changes to software?

"No" doesn't mean Never

Oftentimes, doing nothing is the right thing. Not all proposals deserve implementation. Perhaps there's poor rationale, a lack of clarity, or no compelling use case. Even when changes seem good, not everything will fit into the current long-term goals of the project. In software, stability is also a feature. And backwards compatibility is sacred.

> "In open source no is temporary, and yes is forever" [5].

There can also be benefits to *not* being merged into Core. If a feature is built out using the plugin model, it can simply live on as a community maintained canonical plugin. A plugin will not be restricted by the WordPress release cycle (usually 3 times per year). This extends feedback loops and can prevent faster iteration in the early days of a feature. And while backwards compatibility is still important, it's not applied as a steadfast policy like when code ships in WordPress itself.

But beware of the costs associated with inaction. While inaction does not necessarily equate to inattention, it can contribute to a lack of clarity or risk losing momentum. Time-sensitive windows of opportunity can also be missed, such as supporting an upcoming version of PHP on release day. And postponing necessary fixes frequently makes problems more difficult to resolve in the future.

Deadlines are not arbitrary

"Deadlines are not arbitrary, they're a promise we make to ourselves and our users that helps us rein in the endless possibilities of things that could be a part of every release."

Unless a project is retired, abandoned, or archived, the need to continuously make decisions will always be present. But the reality is that we have to draw a line somewhere at some point. In software, drawing the line usually comes in the form of a planned release cycle and code freezes. "Deadlines are not arbitrary" is another philosophy of the WordPress project that helps contributors to remain practical and focused.

One advantage of being disciplined with schedules and timelines is that it can help reduce the impact of saying "no." Saying no is easier when timelines are clearly communicated and strictly

enforced. When cycles are regular and predictable, the pressure to merge something just because is reduced. There will be another opportunity soon. "Never" becomes "not yet."

Good communication skills are essential for Open Source maintainers. When contributors pour time and effort into a proposal or patch, they deserve transparency. This is especially important when the answer is "no." What aspects of the change look reasonable and acceptable? Where is the rationale unclear or weak? When should they expect a window for reconsideration?

Changing Our Minds

> "In the presence of good rationale, maintainers should be willing to change their mind often" [3].

The best signal that an idea is ready to be reconsidered is the presence of new, clarified, or strengthened rationale. Maintainers should always be willing to change their minds as often as necessary. But they should be confident enough in their conclusions and how they were reached to stand by them under scrutiny. This concept is also referred to as "strong opinions loosely held."

Evaluating Cost and Risk

The most important part of any decision-making framework is evaluating cost and risk. Cost is not just monetary. What is the cost to maintain a given change? What complexities and friction does a change introduce to users? What are the risks for regressions? What are the impacts on extenders? Cost and risk can also be unknown or realized only in the future.

> "All code is presumed harmful, because it will have bugs and maintenance costs, and introduce behaviors that will interact with other features" [3].

Even when one character or line is changed, there is still a non-zero amount of risk. Remember, stability is also a feature, and backwards compatibility is a sacred pact with users that has helped WordPress grow significantly over the last 22 years.

Some Benefits of Backwards Compatibility

In many cases, the project's commitment to backwards compatibility is a sharp tool in the toolbox for limiting the risk of breaking sites.

- There's a safe baseline to evaluate against. "Will this break something that worked before?"
- It discourages reckless refactoring.

- Small, incremental changes are safer, encouraged, and usually preferred
- When plugins and themes depend on past behavior, regressions can be easier to catch early due to a wide range of real-world usage.

Backwards compatibility can also help limit downstream costs such as fewer support tickets, less documentation churn, and a lower level of developer frustration.

Opportunity Cost

Time and resources are finite. Especially in Open Source projects.

Every feature merged or bug fixed is a vote against another that could have taken its place. The time to review, test, document, and support one change subracts time and resources from another somewhere else.

In some situations, a "no" can be given due to an unreasonably high opportunity cost. An example of this can be seen in the WordPress project leading up to the initial release of the new block editor in version 5.0 "Bebo." It was important that as many contributors as possible were focused on the objective at hand. Many changes received a "no" answer in large part because of the amount of resources it drew away from the Gutenberg project.

Maintainers Are the Code They Commit

> "It's easy to write a patch. It's hard to maintain a software project over the long term" [3].

When a change is made to a code base, the committer making that change is taking on a lot of extra responsibility. In some ways, they now own that change and any resulting test failures, bugs, features built on top of the change, or even security issues that may follow. They must be willing to stand behind the changes they make until new rationale is presented.

The code you commit is an extension of you.

Growing Your Community

Growing an Open Source project is not the focus of this essay, but expanding the pool of available contributors should be in the back of your mind with every action we take. Though unique challenges come with growth, a growing project means more resources available to squash bugs and build out features. After all, "given enough eyeballs, all bugs are shallow" [6].

Maintainers play a critical role in community growth by conducting themselves in ways that embrace other contributors. Lead by example in everything you do. Be consistent and approachable. Make space for new contributors by reviewing their patches, answering questions, and

encouraging contributions of all sizes. Recognize that everyone participates at different levels and with different motivations. It may not be immediately apparent how, but every size and shape of contribution is important in some way.

Consider a simple bug ticket with a clear solution. As a maintainer with deep knowledge, you could likely solve this better and faster than a new contributor. But delegating the task to someone else would be more constructive long-term in most cases. The act signals trust, helps build confidence, and strengthens the community dynamics. Other contributors will notice this and be more likely to volunteer or share their experience with colleagues. [7]

A simple code review or "great job" can be the difference between a one-time contributor and a future maintainer. You never know what someone needs to hear, so be generous with feedback.

The Meaning in Our Work

If you've made it this far, you likely care deeply about Open Source software (and if not, you should). Few ideas have reshaped the modern world as profoundly as Open Source. You may not know it, but Open Source is everywhere you look. Routers, refrigerators, trains, cars, rockets[8], stock exchanges[9], and even nuclear research[10].

When you've been trying to reproduce a bug with specific and obscure criteria for over an hour, it's easy to lose sight of the meaning in our work as maintainers. But don't ever forget that your work is important and matters. The "why" will not always be obvious.

The incredible stories of how Open Source is changing the world often go untold. The people writing those stories are busy doing the work, fundraising, caring for others, or simply trying to get by. If you are one of those stories,take the time to share it! Tell a maintainer how their work has impacted and empowered you. If you cross paths with a maintainer of software you rely on and they're looking for sponsorship, support them if you have the means. I promise, they will appreciate it.

Reflecting on Our Principles

We are all stewards of the projects we maintain, championing the guiding principles used to make decisions. No one project, maintainer, or contributor is perfect. Philosophies will be interpreted in different ways at different times, even by the same person. As maintainers, we should reflect on the decision-making frameworks we use.

Are they clearly defined and transparent? Are we following them to the best of our ability? How can our time and effort have a greater impact? Are we upholding the four core freedoms of the GPL? Are we focused on the needs of our long term goals, and the needs of our users?

The ideals I've described above helped WordPress grow to power over 43% of the web. For more than two decades, thousands of contributors have used the project's philosophies to rally behind one shared mission: to democratize publishing.

These principles may not map perfectly to your project or community, and that's okay. Every open source project has its own context, its own challenges, and its own mission. But one truth holds across all of them: every line matters, because software changes lives.

Contact Information

- https://jonathandesrosiers.com/
- https://profiles.wordpress.org/desrosj/

Citations

1 "Usage Statistics and Market Share of Content Management Systems," *W3Techs – Web Technology Surveys*, accessed June 15, 2025, https://w3techs.com/technologies/overview/content_management.

2 Karl Fogel. *Producing Open Source Software*, Chapter 4. https://producingoss.com/en/

3 Havoc Pennington. "Free Software UI." https://ometer.com/features.html

4 Karl Fogel. *Producing Open Source Software*, Chapter 8: Treat Every User as a Potential Participant. https://producingoss.com/en/

5 Aaron Jorbin. *Five lessons from Eight Years as a WordPress Core Committer* https://aaron.jorb.in/five-lessons-from-eight-years-as-a-wordpress-core-committer/

6 Eric S. Raymond. "The Cathedral and the Bazaar," Section 4: Release Early, Release Often. http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html

7 Karl Fogel. *Producing Open Source Software.* Chapter 8: Delegation. https://producingoss.com/en/delegation.html

8 Vaughan-Nichols, Steven J. *From Earth to Orbit with Linux and SpaceX.* ZDNet, June 2, 2020. https://www.zdnet.com/article/from-earth-to-orbit-with-linux-and-spacex/

9 *Red Hat to Ring the NYSE Opening Bell in Celebration of 20 Years of Open Source Leadership.* Red Hat, June 26, 2013. https://www.redhat.com/en/about/press-releases/nyse-0

10 Bahyl, Vladimir, Benjamin Chardi, Jan van Eldik, Ulrich Fuchs, Thorsten Kleinwort, Martin Murth, and Tim Smith. *Installing, Running and Maintaining Large Linux Clusters at CERN.* arXiv preprint cs/0306058, June 2003. https://arxiv.org/abs/cs/0306058

Further Reading

WordPress Project. “Philosophy.” *wordpress.org*. Accessed June 15, 2025. https://wordpress.org/about/philosophy/

WordPress Project. *WordPress Core Handbook – Version Numbering.* WordPress.org. Accessed June 15, 2025. https://make.wordpress.org/core/handbook/about/release-cycle/version-numbering/

Jonathan Desrosiers. “How a Core Committer Thinks: Making Decisions for Millions.” *jonathandesrosiers.com*, June 2025 https://jonathandesrosiers.com/2025/06/how-a-core-committer-thinks-maki

Jonathan Desrosiers. “The Impact of Open Source Work.” *jonathandesrosiers.com*, June 2025 https://jonathandesrosiers.com/2025/06/the-impact-of-open-source-work/

@drmohundro – David Mohundro

github.com/drmohundro
maintaine.rs/drmohundro

I'm David Mohundro and I'm a software architect living in the Memphis, TN area. During the day, I work at Clear Function and at night, I'm helping raise my four kids. I've been blogging at https://mohundro.com and my most well known Open Source project is SWXMLHash.

My early interactions with Open Source were primarily just on the receiving end of Open Source projects - mostly as a user, but also as a learner. I was intimidated by the prospect of working on Open Source and didn't think I had much I could offer; however, the fact that I could look at the source and learn so much from it increased my desire to join in.

All the Open Source projects I've started really began either as learning projects or as tools to help me in my other projects. SWXMLHash started because I was working on an Objective-C project that worked with a SOAP API and I wanted to convert it to Swift as a learning exercise. I barely knew either language, but I knew that working with SOAP APIs by hard-coding XML in a string was less than ideal. By publishing what I had learned in the open, I was able to share with the community. The funny thing now is, I've never released any code that actually *uses* SWXMLHash in it! But others have benefited from it and that is exciting to me.

One of my most exciting moments as an Open Source maintainer was when, out of the blue, a contributor I had never interacted with submitted a pull request to implement custom serialization, which is now the preferred way to work with the library. That's when I realized the library was bigger than me. And that's one of the things I love about Open Source - the projects can take on a life of their own!

Even with all of this, I still have plenty of struggles while working with Open Source - I still struggle with impostor syndrome. I also struggle with growing a regular set of maintainers. Most of my projects are small enough that they don't require more people, but it means the bus factor for my projects is still one.

I haven't had to deal with security concerns *too much* with any projects thus far. That being said, it is still something I have to keep in mind, because the reality of Open Source is that impacts of security are felt far outside of just my code now. I'm not a fan of the term "software supply chain," but it does communicate that the impact of a breach is a lot larger now. Some of the things I try to do are to limit the number of upstream dependencies I pull in - the benefits

are that my code is more self-contained and easier to reason about (and more secure!) as well as simpler to build and release. A harder thing to do for me is to say no to contributions. I haven't had any known attempts to sneak malicious code in a pull request, but I have had requests that didn't really align with what I had hoped to do - those conversations are hard for me to have because I want to be welcoming, but I also don't want to maintain something that doesn't make sense to me or the project.

Another thing that I haven't yet had experience with is a "drive by pull request" that was built entirely with AI tooling, though I suspect it is only a matter of time. I've used enough LLM tooling at this point to recognize the value, while also remaining skeptical, which feels like a healthy balance to me. I suspect we'll see more early Open Source projects that were kickstarted by coding assistants, because the barrier to entry from idea to initial spike is so much smaller... my concern is that as one of these projects sees more widespread usage, we'll see more security issues because the authors will be less familiar with the bulk of the code because it was written by an AI tool.

My advice for anyone interested in contributing to Open Source or starting a new Open Source project - just take that first step. If contributing is too intimidating, pull a project that you respect down and *read* the source. You'll learn a lot from just seeing how others build software. Check out the issues and show some curiosity. If you've got an idea for a project, try building out a spike and see what you can do with it. Finally, if you use a project frequently, let the maintainers know that you appreciate it. Open Source is often a thankless task, so words of encouragement mean a lot!

@fabiocaccamo – Fabio Caccamo

github.com/fabiocaccamo
maintaine.rs/fabiocaccamo

My name is Fabio, and I have been writing code professionally for over two decades. As a developer, I have always been drawn to writing reusable code once and reusing it across projects, refining it over time. Long before I ever heard the term "Open Source", I was already sharing small utilities and libraries with my colleagues at work. It just felt natural. If something helped me do my job better, maybe it could help someone else, too.

Then came **Google Code** (remember that?), and with it, my first real entry into the Open Source world. I started publishing my personal libraries, thinking they might be useful to someone beyond my team. That idea took root and grew fast. When GitHub became the central hub for Open Source, I transitioned there and began sharing `Objective-C` libraries I had developed. The response from the community was immediate and energizing: people were using, improving and talking about the tools I built. That feedback loop hooked me.

What Open Source Means to Me

To me, Open Source is more than just sharing code, it's about learning continuously and staying in sync with the rapid evolution of technology. Compared to when I started coding ~20 years ago, the pace of change today is staggering. Open Source helps me keep up.

It also shapes the way I build things. Knowing that others might use my libraries pushes me to write cleaner, more flexible and reusable code. It's a mindset shift that made me a better developer overall.

Projects that Matter

These days, I focus primarily on `Python` and `Django` (my favorite technologies to work with and the ones I know best). Over the years, I have published around 20 Open Source projects, many of which I still actively maintain.

Here are a few that I'm particularly proud of:

- **python-benedict** — A supercharged `dict` subclass that makes working with dictionaries easier and more powerful.

- **django-admin-interface** — A modern, responsive and customizable UI on top of the default Django admin.
- **FCUUID** — A reliable way to generate and persist UUIDs on iOS devices.

These libraries are used and appreciated by developers all over the world, and I'm incredibly grateful for the feedback and support received.

If you are curious, you can explore the full list of projects I developed and maintain here:

- https://github.com/fabiocaccamo?tab=repositories&q&type=source&language&sort=stargazers

Growing a Community (without Marketing)

I'm not a marketing person, and I don't try to be. Since Open Source is not my primary source of income, I'm not chasing followers or stars. I believe in natural growth. If something I build is genuinely useful, people will find it.

That said, I do share my work occasionally on Reddit and more recently on BlueSky. That's usually enough to get the ball rolling. Over time, those who really benefit from the tools tend to stick around, contribute, or at least say thanks (which is always appreciated).

The Challenge of Maintenance

Maintaining ~20 Open Source projects is not easy, it can be exhausting.

There's a common pattern in Open Source where many people ask for help, features or fixes, but few contribute back, either with code or financial support. That imbalance can be hard to sustain.

To manage this, I've set a few boundaries that help me stay sane:

- **Stay up-to-date:** I keep my libraries compatible with the latest versions of the languages and frameworks they rely on.
- **Zero bugs policy:** I fix issues as soon as possible. I'd rather spend time squashing bugs early than deal with a flood of reports later.
- **Feature development:** I only add new features when I personally need them or when someone sponsors the development.

This approach helps me maintain a high standard while keeping stress under control.

How You Can Support Maintainers

If you're using Open Source libraries, especially in commercial products or at work, please consider sponsoring the maintainers. Your company benefits from these tools, and a small contribution can make a big difference in keeping them alive and maintained.

Contributions don't always have to be financial, though. Bug fixes, documentation improvements, test coverage, and respectful feedback are all useful and appreciated.

The Role of AI

Like many developers today, I've started using AI tools to speed-up my workflow. I use them mostly as assistants: to double-check my code, identify potential issues early or explore alternate solutions.

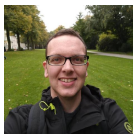
Final Thoughts

For me, Open Source isn't just a hobby or a way to give back. It's part of how I work, learn and grow as a developer. It keeps me sharp, connects me to a wider community, and gives purpose to the libraries I build.

If you're just starting out, my advice is simple: build things that you find useful. Share them. And don't worry too much about marketing or stars. If it solves a real problem, people will notice.

Thanks for reading! :)

@foso – Jens Klingenberg



github.com/foso
maintaine.rs/foso

I had my first contact with Open Source development before I even knew how to program. Back then, I used an Open Source instant messaging client called Miranda IM. While looking for help with the client, I discovered a German online community for Miranda https://miranda-im.de/. The people there were helpful and welcoming, and I wanted to give something back to the community. So I started helping other users by answering their questions. Over time, I got involved in localizing some Miranda plugins, doing QA, and writing tutorials. As a teenager, it was a fascinating feeling to know that something I had contributed to was being used by thousands of people.

As a student, I learned how to use Linux as my daily OS. Since it took me multiple attempts and I knew the common problems beginners face, I organized Linux install parties with fellow students to help others avoid the same issues. It was a fun way to grow the Linux community and get more people excited about Open Source tools.

At that time, I also learned how to program and became interested in Android app development. I started blogging about what I was learning on my website and shared my code samples on GitHub under an Open Source license. One of the biggest advantages while learning was being able to explore other Open Source projects to see how they tackled similar problems. The fact that Android itself is Open Source makes it even easier to dive into the actual source code and understand how things work under the hood.

The two biggest projects I'm maintaining are:

Jetpack Compose Playground

"Community-driven collection of Jetpack Compose example code and tutorials"

https://github.com/Foso/Jetpack-Compose-Playground

When Google introduced Jetpack Compose, a modern toolkit for building native Android UIs, I was immediately drawn to it. From the moment the first public commits landed, I began experimenting, learning, and sharing what I discovered. I created the Compose Playground as a place to collect practical examples, document insights, and help others get started faster.

Because Jetpack Compose is developed in the open, I could follow its evolution closely and learn directly from the source code. A huge benefit for understanding best practices early on. Over time, more people found the project, contributed examples, and even reached out to say it helped them build their first Compose apps.

Ktorfit

"HTTP client generator / KSP plugin for Kotlin Multiplatform"

https://github.com/Foso/Ktorfit

Ktorfit started as a personal workaround. I wanted to use Retrofit-style interfaces, but with Ktor and Kotlin Multiplatform. Nothing like it existed at the time, so I built a simple solution for my own needs. Instead of keeping it private, I decided to publish it as an Open Source project.

By sharing it, I opened the door to feedback, contributions, and real-world use cases I never would have thought of on my own. Developers began filing issues, suggesting features, and even submitting pull requests. Now Ktorfit is helping developers around the world, and that's very rewarding.

One lesson I've learned is that you don't need to release a perfect library to start something valuable. Ktorfit wasn't perfect when it launched. But through the help of others it gets better with every version.

One of the main challenges I still face as a maintainer is balancing my time between reviewing contributions, answering questions, and working on the code base. It's easy to get overwhelmed, especially when the project gains visibility and more users rely on it. Another challenge is setting clear expectations, making sure people know what kind of contributions are welcome, what the road map looks like, and how to get involved. It can be helpful to have good documentation and contribution guidelines you can link contributors to.

In my eyes, Open Source is a shared effort. It shouldn't have to rest on one person's shoulders. The community grows stronger when we support each other. For me, Open Source isn't just about code, it's about learning, teaching, and building things together. I'm excited to keep contributing and hopefully inspiring others to start their own journey in Open Source. Thanks to all people contributing to Open Source, in any kind.

Contact: Jens Klingenberg
https://github.com/foso
https://bsky.app/profile/jensklingenberg.de

@francescobianco – Francesco Bianco

github.com/francescobianco
maintaine.rs/francescobianco

In a world that moves faster every day, finding an anchor—a sense of direction—is not always easy. For me, Open Source has been that anchor, a guiding compass that has shaped not only my career but my philosophy toward technology and collaboration.

My name is Francesco Bianco, and this is the story of how Open Source transformed my life, led me to create a community called Javanile, and inspired me to contribute to various projects, always driven by a single, powerful belief: quality comes through openness and sharing.

The Early Days: Planting the Seeds

Every journey has a beginning. My passion for programming started early with QBasic on my first 386SX computer. As a chess enthusiast, I initially focused on developing chess software before entering the professional world around 2008.

However, my true introduction to Open Source came in Palermo—a city rich in culture and values. It was there that I attended my first Linux User Group (LUG) meetings and hackathons. Though we never formally met, I frequented the same places as Salvatore Sanfilippo, known widely as Antirez, the creator of Redis. That environment profoundly shaped my understanding of Open Source communities and collaborative development.

With numerous ideas bubbling in my mind, I channeled my enthusiasm into creating a local, independent organization called Javanile—a name combining “Java” and “Nile” (the river), reflecting both my technical background and my Mediterranean heritage as a Sicilian.

Building Javanile: More Than Just a Community

Javanile evolved into an experimental laboratory for my Open Source initiatives. Everything I developed for clients was first tested and refined openly through Javanile. The organization’s significance grew when Docker recognized it as a participant in their Open Source program.

A pivotal moment came in August 2016 when Javanile shifted its focus toward DevOps. This strategic pivot defined our primary direction moving forward. Each month, we improve or release

new Open Source development tools with the ambitious goal of enhancing and accelerating software development processes.

This mission continuously motivates us to produce better work. I'm conscious that without the ambition to distribute my work through Open Source channels, it wouldn't have reached its current quality level. Had I remained isolated in my workspace, Javanile wouldn't exist today, and I wouldn't have achieved my current professional standing.

Open Source as a Compass

Throughout this journey, one truth has remained constant: Open Source is more than a methodology; it's a mindset.

For me, Open Source serves as a motivational engine. The prospect of gaining recognition as a contributor guides me like a compass to produce increasingly more software of progressively higher quality.

It teaches humility—accepting that someone, somewhere, might improve your work. It teaches patience—collaboration is slower than working alone, but the results are infinitely richer. It teaches courage—putting your code, your ideas, and sometimes your mistakes out into the world, for everyone to see.

In a way, working in Open Source is like planting trees. You may not enjoy the full shade of your work immediately, but you know that it will grow, benefit others, and contribute to a better ecosystem for everyone.

Expanding Horizons: Contributing Beyond Javanile

While Javanile remains my primary focus, I've expanded my contributions to other significant projects. I'm now a maintainer of BPKG, a formidable package manager. For this opportunity, I must thank the other maintainers who welcomed me and provided invaluable mentorship.

The Maintainer's Challenge

Being a maintainer comes with its unique set of challenges. The most difficult aspect is determining where to invest limited resources. Often, you have minimal time but numerous issues or pull requests to address. The dilemma becomes whether to develop something new and useful or to organize and perform code reviews.

I consider myself a hands-on maintainer who prefers to continue writing code for my projects rather than just managing them. This means that when I dedicate myself to something, I inevitably give it importance and time.

Supporting Maintainers: A Call to Contributors

One way contributors can better support maintainers is by avoiding premature pull requests. Many contributors expect their PR to be merged into a release before they can use it, which creates a paradox. Most licenses allow for distributing modified software, so taking ownership of your fork and distributing it is fundamental.

Only after establishing a usage history for your fork does it make sense to submit a contributive PR. Pull requests with minor changes like punctuation adjustments are less valuable. We should be thinking in terms of "leap PRs" that make significant improvements.

Security in Open Source

In terms of security practices, I've implemented a banner indicating which email to use for reporting vulnerabilities. However, I'm working toward implementing a toolchain based on Software Bill of Materials (SBOM) to enhance our security posture.

The Impact of AI on Open Source

I believe that soon we'll have substantial portions of Open Source code written by AI. We must absolutely strengthen quality gates and continuous integration processes to increase the level of quality control. My answer is to raise the bar even higher.

Looking Forward

Today, my commitment to Open Source is stronger than ever. Through Javanile and through the many ways we contribute every day, we are building not just better software, but a better culture.

As I look to the future, I see endless opportunities: new technologies to explore, new contributors to welcome, new ideas to turn into reality. The road ahead is long, and the work is never done—but that's exactly what makes it beautiful.

Open Source isn't just a chapter of my story; it's the entire narrative thread that ties everything together.

And if you are reading this, maybe it can be part of your story too.

@freak4pc – Shai Mishali

github.com/freak4pc
maintaine.rs/freak4pc

My name is Shai, currently a Staff iOS Engineer @ monday.com. I'm most known as the maintainer of RxSwift and many other frameworks and libraries related to Combine, Apple's first-party Reactive framework. I first met a real-life computer at about 6 years old, when my cousin gave me an old IBM XT for the holiday of Passover. I've been fascinated by this incredible machine ever since, and even experienced first-hand what a "bug" means when my first motherboard burned into flames because of a flying bug finding its way into that same IBM XT.

I grew up in a relatively low-income family and always had to push for myself to learn as much as possible by myself. It started with PHP 3 and HTML in the old days, and as I worked in one of my earlier jobs - my boss had an issue with a contractor that closed shop, gave me a CD with their code and told me - "Congrats Shai, starting today, you're an iOS developer". Little did I know how true he was, as I fell completely in love with this tiny device circa-2011, and was mesmerized with writing code and seeing other people simply touching a screen and interacting with it.

Since then, that has been most of my technical focus (while I still dab with backend and CLI tools), diving deep into Objective-C, and later on Swift itself, and the entire ecosystem.

My reactive journey

When I just started doing iOS, I noticed people are using very cool libraries they found online, and I wondered how I could do the same myself. I started slowly by contributing small bug fixes and later on released a few Objective-C based projects and frameworks, back in the day where not many were available.

From that point I was hooked, and over time as my knowledge grew, I stumbled into RxSwift. I've always been an architecture-fanatic, trying to find the cool new thing and expanding the way I think about code, data flow, etc, and RxSwift tackled an incredible unsolved problem at the time in the iOS community.

I started contributing to RxSwift, initially barely knowing how it works and how the Reactive Extensions standard was defined, and over the following years was able to become a prominent

member of the project, shifting and steering how the project works, modernize it to new Swift standards and make sure it's kept in good shape. In 2017, Krunoslav Zaher (the original creator) had to step away from the project and honored me with taking full ownership of the project. I've officially taken the full journey - from noobie, to contributor, to prominent member, to the primary maintainer of RxSwift and an authority in the field across the Swift community.

What's Open Source for me?

Since I started participating and leading RxSwift, life hasn't been the same, in the best of ways. I've learned what it means to look at the bigger picture of things, what it means to create great code while getting other people involved in a team settings, had the privilege of becoming a world-expert in the topic, writing various successful books, delivering talks in huge conferences, and even landing some great jobs based on this expertise.

So if you ask what Open Source is for me? It's the best way for you to become the best version of yourself. Learn, teach, give back, and grow for yourself as a result of this incredible journey.

It's an incredible way to meet other minds and the wonderful people behind them. I remember the first conference I attended and randomly bumped into contributors from Korea, Germany, Japan, and more. I'd never meet these people otherwise, and it's a huge privilege Open Source enables.

What Makes a Great Contribution (and Contributor) Experience

The best way to start is just caring about a project. If you used it, and it helped you, and you happened to fix an issue, just contribute it back. It will be the start of a beautiful relationship with a community, like it has been for myself. When folks ask me "How can I support the project" - code or documentation updates are most often the best answer!

Growing a community is a relatively tough challenge. Things that help are "Good first issues" and being relatively prompt/available for reviews, but eventually it comes in waves - sometimes there are more contributors, and sometimes less :)

Wrapping up

Open Source has been, and still is, a wonderful journey. I hope to keep doing it for many many years to come, and keep publishing more work even during my day-to-day job.

Feel free to reach out to me on all platforms as @freak4pc (X, GitHub, etc.), I'd love to hear from you!

@hollowaykeanho – (Holloway) Chew Kean Ho

github.com/hollowaykeanho
maintaine.rs/hollowaykeanho

Love using your Open Source and Source Available software? Well, it is May 2025 which is the Open Source Initiative's Maintainer Month! Time to give these neglected but critically important folks a shout out of appreciation. Special thanks to Nick Vidal for giving the opportunity of this collaboration.

I'm (Holloway) Chew, Kean Ho – a maintainer stewarding some fun applications, some serious tools, and my own legal licenses since 2023. Herein lies my story about being a growing maintainer.

This article's writing and artworks are strictly human-created with no Artificial Intelligence's involvement.

> **SIDE-NOTE**: PDF book is now available for free at: https://doi.org/10.5281/zenodo.15334597

Who Am I

I have been active in software development since 2013 and specialized in control computing engineering with 2 First-Class Honours Degrees of Mechatronics Engineering from both Staffordshire University and Asia Pacific University of Innovation. My works cover projects involving proprietary licenses, source available licenses, and Open Source licenses for both non-commercial and commercial goods.

What's Open Source To Me

It's an open commodity of product and processes (analogous to: standardized hammer and nails engineering specifications). The main business objective of Open Source licensing is to make sure everyone can access certain knowledge, product, or processes without any kind of restrictions including monetary restrictions. On the contrary, both Source Available and Proprietary licensing are not free (to change and use) and often associated with monetary charges. Having said that, we need all software licensing types to make the world whole. Source Available licensing and

Proprietary licensing main business objective is to make profits. They all need one another to make the world whole.

However, Open Source licenses must not be confused with Source Available licenses. The latter can get into very nasty business repercussions when treated that way. For case study, Despite Meta marketed its Llama product as "Open Source", its license https://www.llama.com/llama3/license/ is actually a Source Available license where they explicitly stated custom commercial restrictions in §2 instead of the clear freedom offered by Open Source. Using it like any Open Source license without parsing that agreement can allow Meta to pursue litigation easily to stop your commercial pursuit. Always parse the agreement preferably with an attorney.

Therefore, it is important to plan and implement your product with care, achieving the right protection against your business value and yet ensuring non-core software components are declared with the right licenses. Here's a case study: Google primarily profits from digital advertisement so they have chosen to support various great Open Source products like Android operating system, Chromium/Chrome web browsers in order to let the general public have better digital accessibility freely while they can place advertisement lots for running their business.

How Do I Got Involved

My first job was to get involved directly with Linux Kernel drivers from the get-go. That's how I started my journey to Linux and Open Source ecosystems. After my career advancements, I decided to go all-in into Linux and now Debian operating system development and have been working in this domain ever since.

Currently, I am collaborating with folks from FireGiant to upgrade my Source Available automation tool. They provide a MSI packaging solution to my tool so that it can seamlessly package any product into a professional installer for Windows operating system without complicated configurations (MSI installer package is known for its notoriously complicated specifications and executions).

Communities Management

Growing Communities

I'm more of the "Laws of Attractions" kind of person for getting people in. If it solves a business problem, it works seamlessly. I don't actively go around town trying to force people to use it.

My strategy was to set up a friendly environment for people to come in, have some fun, exchange business contacts, etc. closely resembling an evening viking tavern. Folks come in; have some

drinks and meals; dance and sing together to the music; scheduling tomorrow's events together; exchange experiences and stories; and etc. That's the leadership I want to be in Open Source.

Since I got my own AutomataCI automation tool, I generally prefer working with small teams (e.g. 3-4 max) for very effective communications and end-to-end executions. Wherever there can be automated, it must be automated.

Project Involved

At the moment with FireGiant via My AutomataCI end-to-end automation tools:

- https://github.com/chewkeanho/automataci
- (FireGiant) https://github.com/wixtoolset/issues/issues/7896
- (AppImage) https://github.com/ChewKeanHo/AutomataCI/issues/137

Personally, for the general public to enjoy playing My AI Image Upscaler (with no GUI shenanigans):

- https://github.com/hollowaykeanho/Upscaler

For legal licensing, I have my own product licenses since year 2024 mainly to have additional new legal coverages (e.g. data privacy & confidentiality) and a mix of terms and clauses from multiple licenses to meet my generic productization needs:

- https://doi.org/10.5281/zenodo.13770769
- https://doi.org/10.5281/zenodo.13788522
- https://doi.org/10.5281/zenodo.13788522

Challenges as Maintainers

Very limited resources (as in both time and money). I have many visionary projects in my backlog but well, I still have to focus on making a living.

My existing communities are good and I love them! I really wish I can provide them the monetary support they need as a reward for making the world a better place.

Message to Contributors

Not all maintainers are jerks, self-indulgent, or egoistic. Please freely speak out when you're at my "taverns". You can watch Dreamwork's "How to Train Your Dragons" to understand those

emotions, feelings, and cultures. We'll click eventually. Try to frame your queries with curiosity and learning: it always works.

Please do not place immediate or deadline bound schedules. Unless you fully fund the entire project (which can be a huge sum), all of us are actually contributing via our free and hobby times.

Sometimes we do attempt to align our commercial goals for funding to get pitched in. Unless we can turn any stone into gold, well, reality strikes hard.

Security Management

Key Security Practices

Anything that can be systematically automated, I'll do it at any level – No one gets left in the dark be it an intern or juniors; not under my wings.

Security design from start (as early as the first working prototype). In Layman terms, security is a "metal alloy" to be forged from the get-go; not a "Lego" brick composition to be duct taped with.

OWASP https://cheatsheetseries.owasp.org/ and IETF https://datatracker.ietf.org/ are my primary go-to for network security.

Kernel Handbook https://docs.kernel.org/index.html for Linux Kernel.

Debian Handbook https://wiki.debian.org/SecurityManagement & Specs https://www.debian.org/doc/debian-policy/, https://wiki.debian.org/DebianRepository/Format for Debian OS.

Subscribe to every CVE related messaging and mailing list + Gmail filters for inbox auto-organizations. Also subscribe to GitHub CVE Security Advisory or CVE databases.

Comment the Security/Spec reference so the next developer knows where and when it is implemented (searchable using `grep -R "[KEYWORD]"/path/to/directory` command).

"Function before Design" principle - Keep the product minimally functional for minimizing attack surfaces first before starting to apply fancy aesthetics and display.

Biggest Security Threat to Open Source

Threat No.1 is still the deadly supply chain threat. Primarily, I'm looking at the genuine owners themselves or geo-politics "leaders" https://doi.org/10.5281/zenodo.6815012; not the conventional third-party attacking "hacking" entities. Just 1 simple question: "if GitHub pulls GitLab's drastic business policy changes (from `free` to `USD5/user/namespace/month`) or there

is a baseline tariff for network traffic (e.g. from `~USD0.0025/GB egress only` to `USD2 fees for both ingress and egress`), can one mitigate such threat oversight?"; This question alone is suffice to cause enough chaos worldwide. Louis Rossman compiled a lot of toxic business practices for case studying on his YouTube channel https://www.youtube.com/@rossmanngroup/videos.

Threat No.2 is the over reliance on the centralized supply repositories (as in NPM, Python PiP, DockerHub, and Rust's Cargo, and GitHub). If the Geo-politics of the business unit cuts the supply channels off either altering business policies "after the fact" or unfriendly incomprehensible foreign government policy, the entire operation and progress of any ecosystem (Open Source or otherwise) will be badly stifled.

Threat No.3 is younger generations tech influencers spoke too loudly with their too inexperienced "knowledge"; a misguided information. Specialists that I interacted with rarely speak that loud across the industry. A case study is this Shell guide from Google https://google.github.io/styleguide/shellguide.html which completely decelerates the Shell libraries development. Shell and PowerShell are the best candidates for general-purpose automation since they can run without installing anything; not Python that requires its thick interpreter installation; not Maven with its Java runtime requirement. Another case is its recommended function name (`mypackage::my_func(){ ...}`) which is completely not POSIX compatible and only specific to BASH (I believe the writer is lacking POSIX shell experiences and came from a C++ domain). This cost me months of complete libraries rewrite after POSIX realization from cross-running with BSD-based Operating System.

Threat No.4 is the usual resources (funding and etc) shortfall. This is as usual, funding motivates developers to contribute freely. As of this writing, the Open Source Labs (OSL) from Oregon States University is on life-support pleading for funding https://osuosl.org/blog/osl-future/. OSL is currently providing infrastructure hosting for projects such as Drupal, Gentoo Linux, Debian, Fedora, phpBB, OpenID, Buildroot/Busybox, Inkscape, Cinc and many more!

Artificial Intelligence

The Good Side

Large Language Model (LLM) based artificial intelligence (AI) like Claude Sonnet https://claude.ai/new + Deekseek R1 (https://chat.deepseek.com/a/chat/) + Google's Gemini https://aistudio.google.com/prompts/new_chat seriously speed up the materials searches than conventional manual search engines searches. I'm referring to this process:

1. Request both AIs to generate a sample or list of some references for manual searches; AND
2. Analyze their outputs (codes, list, etc); AND

3. Procure the engineering specifications of the tech; AND
4. Construct your own referencing on those outputs and reading those specifications

It also works extremely well with crude language translations (e.g. English multiple localized → languages across the continents).

Stable Diffusion based AI made the world a lot more colorful and took over dull image jobs while letting the human artists focus on important artworks (true story).

Google DeepMind's Udio enables one who has zero experience in music creation but deeply connected to music listening to finally create his/her desirable music (true story).

Convoluted Neural Network (CNN) based NCNN Upscaler finally expands GIMP ability to upscale an image intelligently up to 4x the original per iteration; with different models. Extremely useful for small-sized image archaeological recovery (true story).

CNN-based Text to Speech (coming soon) finally allows Debian OS to speak like human without robotic sounds anymore (true story).

The point: **as long as you create your own version by only referencing it, then the AI seriously accelerates; be it Open Source, Source Available, or even Proprietary licenses**.

The Bad Side

When people misunderstood LLM internal operating functions and misused it for vibe coding or human replacement (e.g. support services).

> Site-note: good luck to those enterprises!

All conventional search engines are completely unusable due to AI generated content poisoning. The era of conventional internet search has come to an end.

It's hard and impossible to publish anything on the Internet without getting AI companies coldbloodily ripping out without respecting local restrictions (as in `robots.txt`).

AI Porn flooded my social media unwillingly.

Biometrics (e.g. face, voice, fingerprint, and retinas) are getting deep-faked way too easily – no longer can tell what's real or not without per-established "Trust on First Use/Meet" and too easy to lose an actual bona-fide identity.

When people think the current LLM (dating 2025) can replace lawyers (coders of the real world) and software developers (coders of the virtual world).

License laundering is still an issue where AI generated content can get as close as the human created version without copyright infringement impact.

My Views

I personally welcome Artificial Intelligence. They seriously empower me by very large magnitudes.

It also reveals our economy and finance's biggest problems "full automation vs. human needs" which is a world economic problem. This is a main issue everyone is worried about but is currently masked by Artificial Intelligence as a threat. That's a story for another day.

To The Future

To all New Maintainers

Congrats! People care about your products. Keep up the good work.

To the aspiring folks...

- Focus on solving a business problem. You'll attract users without needing to roll out a marketing campaign.
- Do not underestimate the "one-time" fun app. It can anytime become a popular tool for everyone to use (true story).
- Always differentiate facts and data from individual opinions. Do not waste too much time on the latter.
- Engineering specifications and actual documentation are way better than "he says she says" stuff by the influencers.

Epilogue

Thanks for reading thoroughly. Well, we come to an end now.

Don't give up and stay connected! My blessing to you.

Contact information

YouTube

- Soundtracks : https://www.youtube.com/@chewkeanho-soundtracks

- Tech Entries: https://www.youtube.com/@chewkeanho-tech-codex
- Personal : https://www.youtube.com/@hollowaykeanho

Independent Research

- ORCID : https://orcid.org/0000-0003-4202-4863
- Zenodo Personal Repository: https://zenodo.org/communities/chewkeanho/

Digital Product Development

- GitHub (Personal) : https://github.com/hollowaykeanho
- GitHub (Business) : https://github.com/orgs/ChewKeanHo/discussions

Mastodon Social

- Main : https://mastodon.online/@hollowaykeanho

BlueSky Social

- Main : https://bsky.app/profile/hollowaykeanho.com

Telegram Messenger

- Main : https://t.me/chewkeanho

Reddit Social

- Main : https://www.reddit.com/user/hollowaykeanho/

Acknowledgements

Special thanks to:

- **Sibert Bronzon (Sweden)** - For always pushing me to grow beyond my comfort zones and explore what's beyond algorithms and control systems.
- **Elvin Ong Boon Leong (Malaysia)** - For always being there when I needed it since our first job together.
- **Julian Hübner (Germany)** - For helping me co-maintaining the AI-driven NCNN application project on the Windows side of support.
- **Cory Galyna (Germany)** - For supporting me all the way throughout my entrepreneurship with her management wisdom.

@hzoo – Henry Zhu

github.com/hzoo
maintaine.rs/hzoo

Interviewer: Congratulations for being part of Maintainer Month! I'm going to ask some questions that I shared with other maintainers. These are basic questions like how you got involved with Open Source, what Open Source means to you.

Henry: I would like to frame how I got involved in Open Source very specifically because I think it's a typical experience. People are very intimidated by Open Source, even though they use it a lot. Consuming Open Source is really easy - you can import a package on NPM or whatever language. But to contribute, you have to make a GitHub account, and while it's easier than before, there are cultural barriers to get around.

Open Source is its own culture, and every project is almost like its own city or country. I think we all want to respect different cultures, and I feel like I would love that people do that for every project or language.

The way I got involved was realizing that somebody was working on the projects that I use. I like to make the analogy that it's similar to volunteering in a garden, or a library, or a church. It's like going to church where they give you breakfast or worship and small groups. These things are set up for you. You participate and then you go home. It's very consumerist.

It's the same with coding - you use React, ESLint, uBlock Origin or whatever, and then one day you might finally realize, "Wait, who is working on this? Who is maintaining this? Are they getting paid?"

Someone told me that contributing is a thing anyone can do. It's very similar to Wikipedia - anyone can contribute, but most people don't. I started by looking into projects I was using personally. You don't want to get involved in a random project - you want something you actually use.

I tried working on a project and it was too complicated, so I ended up working on a linter, which eventually became ESLint. I was fixing spacing and formatting, which is now solved through Prettier. That helped me learn about Babel.

I think most people get involved in Open Source by accident. People are perfectly capable of contributing, but they don't know the non-technical aspects of Open Source.

Interviewer: How did you get involved with Babel?

Henry: I didn't create Babel. Someone else made it. What happened was I just showed up. I answered some questions in the issues and did some basic docs. I didn't even know the project well. I was just interested, and everyone was using it. They put my name on a blog post saying "Thank you for contributing." I felt almost guilty because I barely did anything, but it actually made me want to do more.

Interviewer: That's how you get people involved - you acknowledge them.

Henry: Exactly! You acknowledge them in any way, because everyone's time is valuable. If they spent any amount of time trying to help, that's awesome. It definitely worked - it got me to become the maintainer, which is crazy.

Basically, the person that made Babel left, and I became the accidental maintainer, which is crazy because I didn't even know how it worked. It wasn't my code, but they left.

Interviewer: Were you afraid that suddenly you were maintaining a very complex project? How did you feel about that?

Henry: It was a psychological thing. In my mind, I thought the person that made it would come back. So I considered myself the interim maintainer or interim person in charge. That actually helped me psychologically to not think it was all on me. I thought, "Oh, they're going to come back at any point," and I just kept doing my stuff.

But then a year later, I was like, "Wait, I don't think they're coming back." And that was really scary. But I had spent so much time thinking they were going to come back, and that helped me learn. By the time I realized they weren't coming back, I thought, "I think I can do this."

It's like joining a church, attending one service, and then the pastor leaves and you're the pastor. That's crazy! But it ended up being six or seven years of my life. After a while, I was intentionally like, "Okay, I'm going to do this."

Interviewer: There was a point in your work at Babel that you decided to do less coding and focus more on community. Why did you feel that need? Was it an easy transition or was it by accident as well?

Henry: I wrote about this a little bit in a blog about quitting. It relates to quitting my job too. I worked at Adobe, and I moved to New York City to work there because they found me through Open Source. The only reason I joined was I thought I would do Open Source at work. That didn't happen, but my bosses were amazing. Eventually, I asked them, "Can I do Babel half-time?" and I did that. After a year, I felt like I couldn't compromise myself doing half job, half Open Source. So I just quit, and they were very supportive.

In 2018, Babel was becoming more important in JavaScript, and I felt I could do more to sustain

the project by leaving my job. Having been involved longer, I realized the most important things about the project were the non-technical side - the community.

Yes, it's a compiler, so there are fewer people who can contribute code-wise. But I don't think of myself as that good of a programmer, so I thought, "I'm sure other people will show up to contribute code." And there have been. But not everyone's going to show up to do the other work.

I realized what I thought was important in the project, and no one else was going to do the other work, so I might as well do it because I cared about the project as a whole. It's not like I knew anything about the other things either - I didn't know much about the project to begin with. I was just willing to do whatever was needed.

People showed up to do the code, so I was okay letting others do that while I worked on other things, which was essentially around how to get money, partnering with companies, stuff like that. That took a lot of time and was hard because I'm not a salesperson.

Sales in Open Source is very different because I'm not selling a SaaS product. I'm saying, "You already use Babel every day. Will you give back?" It's weird because everyone's already using it for free. It's more like, "Can you give money so that we can sustain this project?" We could do this for free, but we might burn out. People leave for health reasons, getting older, having kids, getting married, moving. There's no obligation to do any of this, which makes it special but also makes it hard.

Interviewer: This is related to the tragedy of the commons, right? You have 100 companies using the project, and maybe five of them support it. Why should a company sponsor if the other five are already doing that? If you don't have those skills around fundraising and trying to compel people to sponsor, it can be quite challenging.

Henry: Yeah, it's actually crazy that we raised so much. Looking back at our Open Collective, I think we've fundraised over a million dollars, which is actually kind of crazy.

Interviewer: You did a wonderful job. You learned your way.

Interviewer: I think you already touched on ways people can contribute and the main challenges. Would you like to talk about security practices?

Henry: When people come to a project, they feel like they need a very specific thing to do. That's a problem in any organization. People just show up and want to be told what to do. I wish people had more agency - you might make a pull request that might get rejected, but you could communicate with the maintainer: "Is it worth working on?" I think about how we get people to be proactive and have ownership over a project.

Security is similar - it's something that always needs to be thought about at the base level. Most

security practices are practical - you have 2FA, you don't add everybody to the org, you have read-only versus write access.

The supply chain stuff is more interesting. There were times where people shared their old password - I think that happened in ESLint - and then someone published it to npm. Now we have people who might join your project with the intention of doing something bad. I don't think you can solve that because what's the difference between a real contributor who does good work and a fake contributor who's intentionally trying to do something bad later? You can't know that a year later.

I think you have to learn to trust people. That's always going to be hard, especially in JavaScript. When I started, I literally didn't do anything to deserve getting added to the project, but they added me.

Interviewer: When you became a maintainer, there was a lot of trust. It was pretty risky - it could go wrong in so many ways, even unintentionally. If you weren't able to continue the project and deal with the complexities, it could be a challenge. There's a lot of trust happening in Open Source.

Henry: There always has to be a level of trust with humans. I think that relates to AI too - everyone says, "What if we have AI, then we don't need maintainers anymore." But AI can literally create bugs and security vulnerabilities. Someone could probably intentionally insert security issues through AI-generated PRs.

The tragedy of the commons in Open Source is around contribution, not distribution. The whole point of Open Source is that anyone can read and download for free. But the problem is there aren't enough people helping.

Nadia Eghbal, who I did a podcast with, wrote a book called "Working in Public." She has this two-by-two grid of different kinds of projects based on contributor growth and user growth. Babel was cited as an example of what's called "the stadium" - low contributor growth, high user growth. So you have like two or three people and hundreds of thousands of users. That just doesn't work.

The problem with AI is that it could potentially make this problem worse. You might think, "We're going to have more contributors because it's easier to contribute." But I'm thinking, well, someone has to review it. Yes, you can use AI to review the code, but I don't think anyone would use a project where the reviews were AI-driven. The person still has to look line by line, just in case of a supply chain attack.

I also had a thought that AI might change what things would be Open Source. I feel like it will lead to fewer small projects being Open Source because AI could just copy them. I could give the AI this project, and it would just inline it in your project. Why even import it from a

package manager when AI could just implement the code? It's basically like forking, but you don't need a repo - it's just inline in your project.

So I feel like it would lead to bigger Open Source projects because AI will use them as dependencies, but it won't literally rewrite them. But for all these small things, AI will just rewrite them automatically, so there's almost no need to have small projects.

Interviewer: Is there something else you'd like to share?

Henry: I would love to share why I do Open Source and why I quit my job. The podcast that I do, it's called Hope In Source. That has been really fun for me.

I would like to offer to people in Open Source to think about it in a spiritual sense. There is something that people could be missing out on by not thinking from that lens. When I think about Open Source, I think about volunteer work, helping people, and that's very similar to serving in a church.

For me, that's always been the motivating factor. People ask, "Why did you quit your job? Why do you do this?" I think it's a great way for me to live out what I believe - values around ownership, community, stuff like that. It's different from the default way of thinking about it in Silicon Valley.

I want to not just say I love community all the time, but live that out in my life. That was one of the reasons why I quit - how do I best live out these values? I also felt that if I stayed at a company in that culture, it would change who I am. I want to discover new values or retain the old values that I've had.

Interviewer: Thank you so much for sharing your thoughts, Henry. I really appreciate your time to share your experience with the community.

Henry: Thank you! A shout out to the current maintainers: Nicolò Ribaudo, Huáng Jùnliàng, liuxingbaoyu.

@jamietanna – Jamie Tanna

github.com/jamietanna
maintaine.rs/jamietanna

My name's Jamie Tanna and I've been contributing to Open Source for >11 years, which extends to before my professional career began.

As with many others, my first foray into Open Source was a few spelling and grammar fixes, but I remember being part of a computer security forum several years prior, where we would share code.

Over the years, I've contributed in various ways to projects - helping answer others' issues on issue trackers, submitting suggestions for bug fixes, implementing features, writing/improving documentation and helping contribute to discussions about the way the project is shaped, but I've also been one of those annoying "any updates?" users, too.

As I started more regularly contributing to Open Source while I was working at a large financial institution, I was incentivised to contribute to projects out-of-hours, instead of going through our fairly gnarly Open Source policy.

This was a bit frustrating as it slowed down my ability to help fix issues that we were blocked by internally, but it still at least meant I was getting the work done. On the flip side, the company missed out on having their corporate email address associated with the changes.

In recent years, I've fortunately worked at companies with a better policy around contributing to projects, but I've also found that I'm a much stronger advocate for myself (and for the company) to allow me to do the Open Source work on my work time, rather than my personal time.

I've enjoyed being in the place in my career and at a company where - if there is a need to raise a bug upstream, or indeed finding the fix to a bug we're facing, or there's functionality we want to contribute, I can go ahead and do that without much pomp and ceremony.

At the end of the day, we all benefit from continuing to work on the commons, companies most of all!

Collector of Open Source projects

I have quite a number of projects I've built (with differing levels of support) that I have a page on my website listing all the projects!

The main ones that folks may know are `oapi-codegen`, the OpenAPI-to-Go code generator, and dependency-management-data, a tool for better understanding your dependency tree.

I've also previously been the primary maintainer for the Jenkins job-dsl-plugin, and on the maintainer team for Wiremock, and I've had contributor access to several projects over the last decade, and I'm also a fairly regular contributor to a number of other projects like Renovate.

Something that may be clear from the projects is that they're in a spread of different ecosystems - there's a lot in Jenkins, Java and Cucumber, or a number in Go, or around the OpenAPI ecosystem, or more recently, in the dependency management and insight space.

These all follow where my "focus" as an engineer has been at a given point in time, some of which I've been continuing to work on after I've stepped away from that ecosystem.

The Open Source project I've been most consistently contributing to is my personal website and blog, which is also an Open Source project in of itself. It's shared freely to the world, with mostly `Apache-2.0` code snippets, and I truly do see this as another project I continue to invest in, whether writing blog posts as a form of blogumentation, or sharing some lukewarm takes about Git commits.

Where does the time go?

As a maintainer, I'd say the biggest challenge I have right now as a maintainer is *time*.

As above, you've seen that I have a number of projects - all of which are in various states of maintenance needs - and my blog where I want to actively continue working on, as well as "life stuff".

Of my two "main" projects, `oapi-codegen` demands a fair bit of time, and dependency-management-data is something I'm continuing to build in functionality for.

At time of writing, `oapi-codegen` has 503 open issues, and 173 open Pull Requests. Earlier this year we were up at ~600 issues, and ~150 open PRs, as far as I can remember.

Although I describe myself as co-maintainer of `oapi-codegen`, my co-maintainer (and project creator) is currently super busy and unable to work on the project. This means I'm doing the majority of the work - there's a lot of work to triage, prioritise, debug, add test cases to reproduce and then fix issues, let alone taking on new functionality that could be seen as "it's *just* a case of merging this PR".

As noted in a post from last May, we indicated that we'd like to try and move to a more sustainable model, taking in sponsorship to try and make the work on the project more consistent.

I'm incredibly fortunate to say that I have several sponsors who are great, and each paying for 1hr/month for me to work on the project, and my employer, Elastic gives me 4hr/month to work on the project.

I understand that this *absolutely* isn't the norm and is really quite privileged to be able to say I not only have some "free time" to work on the project, let alone that some of it is paid! With the appreciation I hold for their sponsorship to pay for a number of hours of work, there is still, unfortunately, not enough time.

If you rely on any of the projects I maintain, I'd love to talk with you about sponsoring me!

The art of "doing it right"

The other, very intertwined, difficulty I face as a maintainer is complexity.

As `oapi-codegen` is built on top of OpenAPI, there's a lot of functionality that a user *could* use in their OpenAPI documentation, which we then need to support in `oapi-codegen` - it's a very powerful specification that allows elegant descriptions of metadata, but then requires tooling to respond to those complex use-cases.

To make matters a little harder, we often need to be led by users with examples of functionality they're using, before we can add support into the project for what they're doing.

When we end up trying to fix these sorts of issues, we also want to support this in a backwards-compatible way, making sure generated code only changes if necessary, otherwise making it an opt-in feature. This adds complexity with more internal feature flags, on top of complex specifications that may be in use.

As a project, we're trying hard to build sustainability in our documentation and our test suite - I recently spent a lot of time rebuilding our documentation, which now means that previously raised contributions (some as old as ~6 years) need to have this documentation retrofitted to them (usually by us, the maintainers) before they can be accepted.

As we're trying to make the project more sustainable - with better documentation and test suite - this does come at the cost of having to slightly slow down to make sure changes are done in the "right" way.

One option is I can already hear folks saying is that we could "use AI to triage those issues" or "use AI to suggest specifications that need to be implemented", but I worry about the accuracy and I'd prefer to have issues taking a bit longer to get to, rather than being solved incorrectly, especially as I'd prefer to personally introduce a bug rather than it being via i.e. an LLM.

Advice to maintainers

If I had any advice for any new or current maintainers, it would be learn what your boundaries are, and enforce them.

This is a good life lesson, too, but especially with Open Source where we see a lot more entitlement, a lot of the time from large companies.

In Open Source, we're constantly doing free work which the industry benefits from, and as Mike McQuaid puts it, Open Source Maintainers Owe You Nothing.

Understand what this project is *to you* the maintainer and what you do/do not want to take on as contributions, and then ideally make it clear by using something like a GitHub badge for unmaintained.tech or Joseph Hale's PSAs to indicate maintenance status, or adding information into `CONTRIBUTING.md` about what is/isn't in scope for the project.

Unless you're getting paid as your day job, it's taking up precious free time, and although we love you for doing it, you should only be doing it if you enjoy it. And even if you are getting paid for it, you can't magic up more hours in the day, or necessarily prioritise things differently based on the whims of *one* of your users.

Secondly, I'd indicate that you should try to be transparent and intentional.

In a post to `oapi-codegen` last year, we made it clear that we appreciate there's a gap in the maintenance we've been doing, and do want to improve it, but can't feasibly without more financial support.

Having this difficult conversation with your users can help set expectations, and I promise you it'll make you feel better.

Call to action for users

I've got a couple of final things I'd like to leave you, dear reader, with.

The first one is to go forth with empathy, and to remember that your maintainers are *people*. We have lives behind the screen you may not be aware of, other commitments that may mean this project we work on *for free* and for the good of others isn't the most important thing.

The second is to please go and say something nice to one of the maintainers in your life. I guarantee they may only get 1-2 positive messages all year, or it's usually "Hey, this project is great, thanks! BTW, there's a massive bug ..." and that it'll make their day.

And a final one is to talk to your company about sponsoring Open Source, and see if you can start helping the people doing the very hard work you rely upon.

Let's try and make the human side of Open Source more sustainable.

@jbednar – James A. Bednar

github.com/jbednar
maintaine.rs/jbednar

I am the original founder of the HoloViz.org project, which provides open-source Python libraries for scientific, engineering, and analytical data exploration and visualization, PyViz.org, which catalogs all the open visualization tools available for Python, and Pandata.pydata.org, which provides a curated set of highly scalable data processing tools.

I have been a vocal proponent of Open Source since grad school in the 1990s, where I embraced the new (and still quite rough!) Linux operating system and GCC compiler. My goal was to ensure that my research software would be reproducible by anyone, without any restrictions. Approximately fifteen people ever tried to use any of that esoteric brain simulation code, but I kept focusing on open science when those around me were locked into proprietary tools that I felt were not worth that loss of freedom.

Once I became a computer-science professor in 2004, I embraced Python and taught courses arguing that Open Source and flexible "scripting languages" like Python were the way we could finally achieve the still-elusive goal of software reuse (instead of starting every new system from scratch!). However, I felt like a curmudgeon shaking his fist at the sky, with those fanciful ideals seemingly impossibly far away from reality.

Two decades later, nearly every cloud computer runs Linux, and Python is now the most popular computing language in the world. Instead of starting from scratch, it is now the norm to stitch a large collection of libraries into a complete Python or Javascript system, requiring very little new code. Open Source won, and software reuse is real! I feel privileged to have seen such a profound shift in how software is written and used. Today's scientists and engineers no longer have to choose between a proprietary platform with hard-coded functionality, or else writing entire systems from scratch in a language like C (like I had to do as a young researcher).

My own group's open-source tools started out modest, with Param.HoloViz.org starting in 2003 to help make our brain simulator have modular, well-defined software interfaces, and HoloViews.org joining it in 2010 to make it easier to use Matplotlib for complex multi-dimensional data. Our tools were easy to distribute since they were pure Python, but they needed to be combined with C or Fortran libraries like NumPy to do any real work, which limited their use to old coders like myself who could compile those languages. Anaconda.com started to change that in 2012, providing pre-compiled libraries that allowed numerical computing in Python to really take off.

By 2015 I had left academia along with three of my former Ph.D. students and we all joined Anaconda's consulting group, which let us focus on our open-source tools nearly full time, instead of working on them when they were meant to be writing papers and grant proposals. For each customer, we now walk in with deep expertise on Python's open-source tools, offering to quickly put together solutions reusing existing libraries to save them time and money. We then pitch the customer on new open-source capabilities that we propose to add to the open tooling so that they can use it without needing to maintain the tools, and so that they can later take advantage of advances funded by other customers. In client project after client project, we have steadily expanded our tools to include Panel.HoloViz.org for building web apps in Python, Datashader.org for rendering enormous datasets, Colorcet.HoloViz.org for perceptually accurate colormaps, hvPlot.HoloViz.org for quickly exploring data, GeoViews.org for working with Earth-centered data, and now Lumen.HoloViz.org for exploring data using natural human language thanks to new Generative Artificial Intelligence (GenAI) large language models (LLMs).

Somewhere in there we introduced the name HoloViz to group these libraries together and show that they are all maintained as a group, and created HoloViz.org and Examples.HoloViz.org to show how to use them all together. In the meantime, the Python software ecosystem had become so diverse and complex that we needed to create PyViz.org as an unopinionated overview of what became *hundreds* of visualization libraries. The dashboarding and web applications section of PyViz.org went from four tools at the beginning of PyViz to now *dozens* of competing libraries. In my career, we have gone from having no software available to reuse, to having software but having to struggle to build and connect it together, to today's problems of having to choose between so many confusing options.

Along the way, our tools ended up having a real impact, far beyond our original academic community. Param was used only in our brain simulation framework for its first ten years of life, but it is now downloaded from PyPI and anaconda.org over a million times each month. All told, the HoloViz tools are downloaded 10 million times a month, compared to maybe a single thousand people who have ever read my academic books or papers. Open Source has had a much more profound impact than just about anything else I have done in my professional career. What a journey it has been!

Thanks to Philipp Rudiger and Jean-Luc Stevens (the original authors of Panel and HoloViews), Jefferson Provost and Chris Ball (the original authors of Param), and all the many other members of the HoloViz team past and present.

@jcubic – Jakub T. Jankiewicz

github.com/jcubic
maintaine.rs/jcubic

My story with Open Source started when I finished high school in 2000. I started using GNU/Linux around that time. At first, I used a dual boot with Windows, but soon ditched Windows and started using Linux exclusively.

When I started college in 2002, I read two books: *Hackers* by Steven Levy and *Free as in Freedom* about Richard Stallman and the GNU project. Those books had a huge influence on me. So, my introduction to Open Source came from Free Software and Software Freedom. Note that the term hacker means clever programmer, and not a cybercriminal.

My first contact with Open content was in 2006 when I joined OpenClipart. It's a site with Public Domain vector graphics in SVG format. SVG is the standard vector format for the web.

In 2007, during my last year of college, I had severe mental health issues, and the medication I was taking made me unable to work. I also broke my laptop. I was not able to get my degree because I was too sick to write my final thesis, which was the only thing left to do. Around 2010, I more or less recovered (I still needed to take my meds), and around that time, I got myself a new laptop. You can say that this was the time when my digital life on the Internet began.

My first contribution to Open Source was reporting issues on GitHub to a project called BiwaScheme, around 2010. I also contributed by updating documentation. It first started as a blog post and later ended up as part of the Wiki.

At that time, I also created my first bigger OSS project called jQuery Terminal, which I'm still maintaining. Back then, jQuery was the most popular front-end library. Because I was inspired by RMS, I picked the GNU GPL license for the project. Soon, I was asked to make the license more permissive, so I switched to GNU LGPL. I got a few contributors, and then I was asked again to change the license to MIT or to dual-license, like jQuery did back then. This is what I did. After a while, I was approached by a company that wanted to use my project but asked if all contributors had given permission to change the license. So, I needed to contact individual contributors and ask for permission. From that time, I started using the MIT license for JavaScript projects that run in the browser.

The same year, I also joined as a developer of OpenClipart and AikiFramework that OCAL used. Aiki was an Open Source project led by Bassel Kartabil that was later killed by the Syrian

Regime during the civil war.

During all those years, I created a few Open Source projects that were somewhat successful, like Wayne, Sysend, Tagger, LIPS Scheme, Gaiman, and chat-gpt Bookmarklet.

But my biggest project that I'm a maintainer of is isomorphic-git. It's a pure JavaScript implementation of a git version control client. Isomorphic means that the project works in both the browser and Node.js. The aim of the project is to be as close to "canonical" git as possible.

The story of joining isomorphic-git started when I created a small application called Git Web Terminal, where I used both isomorphic-git and jQuery Terminal. When working on Git Web Terminal, I started getting involved with isomorphic-git on its GitHub issues, asking questions, providing answers, and writing some code. Soon, the author of the project asked if I wanted to join the isomorphic-git organization. After a while, the original author left the project, so I felt obligated to maintain it. I didn't want the project to die, like many others out there. My code contribution was minimal, but I did review and merge Pull Requests and kept the project alive. I still maintain it to this day.

While I was asked to write my Open Source story, I was also asked a few questions that I will answer below:

What's Open Source to you?

For me, FOSS is mostly about helping other people. You create something that you want to share with the world. It's the same kind of volunteer work as contributing to Wikipedia, which also has an impact on a lot of people.

How do you grow your community?

A community of contributors starts from normal users that you need to get first. So, I would start with investing in marketing. There are a lot of things you can do to promote your project, but in the long term, I think that the most important is SEO. I've even written a blog post about this:

- How to Promote Your Open Source Project with SEO

What are the main challenges you face as a maintainer?

I think the biggest challenge is getting people to contribute meaningful changes to my projects. And getting bug reports when something is not working. From my experience, the majority of

users don't report bugs. If something is not working, they just deal with it or use something else.

But I've noticed that users report bugs if they are asked to. Isomorphic-git, which I'm a maintainer of, has an internal error that asks users to report a bug. Because of this, we had a flood of bug reports related to Salesforce CLI. It was their fault, and they quickly fixed the issue, but it took a while before new bug reports stopped being created.

What advice would you give to current and new maintainers?

Invest in automation, and always reply to users' feature requests and bug reports, even if it's just:

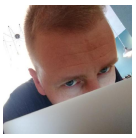
Thank you for your report.

What do you think are the biggest security challenges facing Open Source today?

I think the biggest challenge may be trusting tools like AI too much. AI was trained on code found on the internet, like StackOverflow. And StackOverflow is known for having code snippets that are vulnerable. If you trust AI-generated code, you may accidentally introduce vulnerabilities in your code. The most problematic is so-called vibe coding, when you don't even check what the AI has produced.

If you're interested in what I do, you can find me on Twitter/X at @jcubic.

@jugmac00 – Jürgen Gmach



github.com/jugmac00
maintaine.rs/jugmac00

Hi, my name is Jürgen Gmach, and I am an open-source advocate by heart.

Although I started writing code already at the age of 13 (generation C64), and I have been a professional software engineer since 2007, I only started contributing to open-source software a couple of years ago.

While I had been using many open-source applications and tools like Drupal, WordPress, Roundcube, Dovecot, Postfix, Apache, Nginx, PHP, Python, and a few Linux distributions for several years, I never felt the need or the urge to contribute to them - they just worked.

Turning point

Things changed when I became the maintainer of a 14-year-old custom intranet application built on the Zope / Python 2 stack. Python 2 support ended in 2020, which meant I needed to migrate the application to Python 3. The problem was that there was no Python 3 version available for Zope.

As a team of one, migrating both the application and the Zope framework was a mission impossible.

... But Zope is open-source software

Luckily, Zope is both open-source software and was still used by a couple of companies, which funded a series of sprints, where dozens of developers from diverse companies and organizations came together and with combined efforts over several years, successfully ported Zope to Python 3. An open-source success story!

This was only the beginning

During the migration, we needed to make sure that Zope is both working on Python 2 and 3 at the same time. As we had to migrate around 300 libraries, it was a real pain to create and update separate environments for Python 2 and 3 for each of them.

During the sprint, I learned about a tool that automatically creates different environments for your libraries and runs the test suite for each configuration. The tool is called tox, and I immediately fell in love with it.

tox (always lowercase, even at the beginning of a sentence) had a few issues and rough edges, so at first I created several issues on its bug tracker, but after a while, I felt this relentless urge to give something back. At first I started looking through all the reported issues of the project, to get an overview of what needed to be done, closed a couple of already fixed issues, and then I just started working on tickets - one after another. The code base used bleeding-edge tools and Python versions, and I got invaluable feedback on my pull requests from a very experienced engineer. I just could not stop anymore.

Congratulations

After a while, the maintainer of tox announced me as a maintainer myself!

I could not believe it, becoming the maintainer of a tool, which offers so much value, which gets downloaded more than 20 million times a month, and which is used both across other high-profile open-source projects and multi-billion dollar companies alike.

Being the maintainer of such a high-profile tool might have also had a big impact when I successfully applied at Canonical. Meanwhile, I work full time on Open Source helping to maintain Ubuntu, one of the most popular Linux distributions of our time. I also contributed to several hundreds of open source projects to a varying degree, and I am a frequent speaker at conferences, where I share my experience and try to remove the barriers for others to become part of the Open Source success story.

Maintainer woes

Being a maintainer of a high-profile open-source application such as tox is not always shiny. You need to manage your time and the expectations of your users.

There is always the decision of whether I should fix the latest reported bug, or spend time with my family.

Users of your open-source applications are humans, too, and as such there are also not so friendly ones. Several times I faced angry users who complained about a bug, or who insisted heavily on new features or requested help to debug their 500 lines configuration file for their multi-billion dollar company.

At this time, I read the blog post I am not your supplier, which resonated with me and helped me prioritize the important things in life.

Free lunch is over

While you can handle overambitious requests more easily, it is another thing to receive a security issue - especially when you are busy in life with other things.

Then you need to have the ability to defer work to other maintainers and ask for help in the community.

Security

When it comes to security, we both apply the best practices like two-factor authentication, using security linters, and so on, at some point you need to rely on others.

For tox we make heavy use of the security features offered by GitHub, that is e.g. Dependabot which helps with keeping the dependencies secure, which is one of the greatest challenges in today's security landscape. Supply-chain attacks are a real threat.

What you can do

I love being an open-source maintainer, getting in touch with users across the world, and knowing my contribution helps tens of thousands of users.

As an open-source user, you can do a couple of things. The most obvious one is certainly helping to fund the project, either directly or via your employer which relies on the project. Even the smallest contribution counts.

But there are many other ways. Take great care when creating issues, e.g. making it as easy as possible for the maintainers to reproduce a bug.

Even if you are a happy user, and you do not need a bug fix or a new feature, reach out to the maintainer of your favorite project and say "Thank you!". This will conjure a smile on the maintainer's face - I know firsthand.

Please never forget, that even with all the anonymity the web offers, at the very end, a maintainer is still a human. Be kind. Be friendly. Be part of the open-source movement.

@jviotti – Juan Cruz Viotti

github.com/jviotti
maintaine.rs/jviotti

Hey there! I'm Juan Cruz Viotti. I'm a member of the JSON Schema Technical Steering Committee, O'Reilly author (Unifying Business Data and Code: Designing Data Products with JSON Schema), award-winning University of Oxford alumnus, and founder of Sourcemeta, my tiny company where I work in open source for a living.

Some of my notable Open Source current work includes Blaze, a high-performance C++ JSON Schema validator, a JSON Schema CLI designed for maintaining large JSON Schema ontologies, the Learn JSON Schema popular documentation site, and JSON BinPack, an on-going research project for space-efficient IoT data transfer. I also help co-organising the JSON Schema track within the API Days Conference and I've been a mentor in Google Summer of Code a couple of times now.

You may also know me as the original author of Etcher, plus I've been casually involved in many other projects in the past, such as helping architect the Node.js Single Executable Application initiative.

It all started with Open Source

Many professional software engineers have asked me for advice on how to get involved in Open Source. But for me, it was the other way around: Open Source is what gave me a career.

As a kid growing up in Argentina, I loved building things and tearing them apart. Of course, it was no different when we got our first family computer. After overly fiddling with it and making it unusable for others several times, I got my own, which only made the fiddling problem a lot worse.

Computer magazines and the Internet quickly introduced me to GNU/Linux and the Open Source philosophy. Back then, Ubuntu was gaining attention and Canonical was mailing live CDs for free to boost adoption. I never thought it would make it through customs to a 12 years old kid, but it did. I remember having to reassure my mother that it was nothing dodgy and that I had not somehow spent any money on it without her permission.

It is hard to use GNU/Linux without being exposed to source code, which got me excited about programming. For years, I was hanging out at IRC servers trying to send patches to arbitrary projects and reading every programming book I could get my hands on. Most of those books were from O'Reilly, and seeing my name on an O'Reilly cover now always gives me a ton of flashbacks.

When I was around 16 years old, GitHub was gaining popularity. I joined early on and started publishing a lot of random projects, ranging from C experiments with UNIX sockets to my own Linux package manager. To my surprise, people reached out and I landed a few contracting gigs with some overseas startups. As a 17 years old teen, I couldn't take international wire transfers under my name, so I took Bitcoin (and made good money when Bitcoin exploded some time later!)

Soon after I turned 18, I opened a bank account and landed my first proper full-time job as an adult at a tiny London-based company that would eventually become Balena, a well known multinational IoT startup valued at $558M with $101M in funding. It was a hell of a roller coaster (in a positive way!), and my experience there serving as an Engineering Lead taught me many valuable lessons. Of course, Balena is almost entirely open source.

From Open Source hacker to founder

Thanks to Balena, I eventually moved to England, and as a Harry Potter fan (who read the series too many times), I was thrilled when I was accepted to the University of Oxford. I spent my time there studying formal methods, mathematical proofs, formal specifications like the Z Notation, and formal concurrency models like CSP. Hint: JSON Schema is not too different from these things!

By the end of my degree, my dissertation focused on the use of expressive schema languages to derive binary serialisation rules for higher compression when transmitting telemetry data over 5G and satellite. The result was JSON BinPack, an experimental open-source binary serialisation format based in JSON Schema that proved to be more efficient than every other tested alternative in every single tested case. I was awarded the CAR Hoare Prize for the best dissertation and resulting project.

While wrapping up at the University of Oxford, I started contracting at Postman, a popular company in the API space. In a twist of serendipity, Postman started employing JSON Schema core contributors as a way to sponsor their work. After my previous research around JSON Schema, this fueled my involvement there even more, and I eventually joined the JSON Schema Technical Steering Committee.

My official JSON Schema membership and my recently published O'Reilly book touching on

JSON Schema (Unifying Business Data and Code: Designing Data Products with JSON Schema) led me to various engagements with companies that were part of the community. This gave me additional firsthand experience on how substandard the ecosystem was for use cases beyond trivial.

For example, validating a simple configuration file is easy. But properly pulling off a large-scale schema ontology or API Governance program was borderline impossible from a tooling point of view. Educational resources were lacking too. Before I kickstarted Learn JSON Schema, there was no reference documentation for JSON Schema at all. You had to read and grok the specifications, which was no easy task.

I saw this gap as a great opportunity to capitalize on my own self-funded company: Sourcemeta. Balena taught me how customers can see Open Source as a competitive advantage, and of course, I embraced the same principle at Sourcemeta.

The future is Open Source

JSON Schema is one of the most ubiquitous technologies I saw in a while. It is everywhere, from Fortune 100 companies and scientific institutions all the way to governments and early stage tech startups.

At Sourcemeta, our goal is simple: we want to provide the very best enterprise-grade tooling and services across the entire JSON Schema ecosystem. All our offerings, including the fastest schema validator available in the market and an upcoming self-hostable schema registry, are publicly available on GitHub under the AGPL-3.0 and a non-copy-left licensing model that guarantees our sustainability.

JSON Schema itself is Open Source. Sourcemeta offerings not only help our users get more out of JSON Schema, but also help fund my own continued work in the JSON Schema Open Source organisation. It works out because they both feed into each other.

Finally, I believe that Open Source is not a philosophy. It is what makes software powerful, and that is why we need more people working in Open Source. However, donations is not a sustainable model and we don't do ourselves a favour by portraying Open Source as charity. Instead, I encourage entrepreneurs to build commercial Open Source products, and users to prefer buying from open source vendors. Together, we can create a market where Open Source is the rule, and not the exception.

After all, I wouldn't be here if it wasn't for Open Source.

- Personal website: https://www.jviotti.com
- LinkedIn: https://www.linkedin.com/in/jviotti/

- GitHub: https://github.com/jviotti
- O'Reilly Book: Unifying Business, Data, and Code: Designing Data Products with JSON Schema

@karlhorky – Karl Horky

github.com/karlhorky
maintaine.rs/karlhorky

Hi, I'm Karl Horky (GitHub, LinkedIn), Technical Founder at UpLeveled - tech education programs for all skill levels.

In an educational landscape of AI-generated solutions, disconnected islands of knowledge and barriers to entry, I focus on helping students level up by designing accessible curricula and contributing to Open Source.

Then and Now

I've been in Open Source for over 13 years, and in tech for more than 20, through which I have used a range of languages and technologies, from QBasic and C to Perl, PHP, Python, Ruby on Rails and then finally on to JavaScript / TypeScript.

Now I mostly work with TypeScript, React, Next.js, Node.js, SQL and Bash.

Through my work in education, I've become interested in:

- new approaches in web frameworks like React Server Components / Server Actions and the Islands Architecture
- SQL-in-JS tooling like SafeQL, `prettier-plugin-embed`, Postgres.js
- secure-by-default and pit-of-success approaches to building safe and correct software, eg. enforcement and guidance through linting rules and expressive, strict API design
- patterns to reduce abstraction and indirection in code

UpLeveled and Open Source

My work at UpLeveled has focused on designing, developing and delivering accessible curricula for students ranging from beginners to more experienced engineers.

As part of this work, we maintain some of our own Open Source projects:

- Preflight: command line interface for students to check their code quality

- `eslint-config-upleveled` and `eslint-plugin-upleveled`: ESLint config and plugin with custom rules
- System Setup: Windows, macOS and Linux setup guides
- numerous example repositories like Examples of Broken Security with Next.js + Postgres.js and UpLeveled Next.js example - Winter 2025

Papercuts

In addition to our own projects, UpLeveled also lives what we teach and aims to be good Open Source citizens by contributing to other projects when we encounter problems.

One common type of problem we encounter is the "papercut":

1. bugs or inconsistencies which appear to be minor
2. to more experienced developers, mildly annoying and not worth fixing
3. to students, potentially a blocker to their learning

These papercuts can include errors during setup steps, documentation issues, small bugs and even security issues.

By fixing these papercuts, we can help students focus on learning and building projects, and raise all boats by contributing the fix to everyone else.

Upgrading

Another area of focus for UpLeveled is keeping up to date:

- we produce new versions of our curricula multiple times per year
- we upgrade to new versions of OSes, browsers, runtimes, frameworks and libraries
- we report and fix any issues we encounter during the upgrades
- we adopt new patterns and consider new tools

Some examples of issues and pull requests related to these upgrades:

In January 2023, while adopting the Next.js App Router and switching our material to React Server Components, we found that Route Handlers did not have the same capabilities to check return types using TypeScript, and contributed this feature to Next.js:

- Add optional generic parameter to `NextResponse` in `vercel/next.js`

During a June 2024 iteration on our Expo / React Native lecture, we switched the scaffolder and template we used and dropped the obsolete config in `.npmrc`:

- Switch to `create-expo-app` + `blank-typescript`, remove `.npmrc` cmds in `upleveled/system-setup`

More recently, a March 2025 upgrade to `eslint-import-resolver-typescript@4.2.0` caused resolution errors for Bun modules like `bun:test` while using `eslint-plugin-import-x`, which we fixed with a documentation update:

- Document eslint-import-resolver-typescript `bun` option, fix ESM import in `un-ts/eslint-plugin-import-x`

Supporting Ecosystem Evolution

Over the long term, another goal of UpLeveled is to help evolve the ecosystem by extending compatibility, encouraging adoption of new technologies and discussing new standards proposals.

Extending compatibility has included issues and pull requests such as:

- Node.js Type Stripping in `node_modules/*/*.ts` in `nodejs/typescript`
- Add nested transforms in `porsager/postgres`
- Support for SafeQL on Windows in `ts-safeql/safeql`
- Recognize referential actions as keywords in ON UPDATE/DELETE in `sql-formatter-org/sql-formatter`

Encouraging adoption of new technologies has also ranged across multiple topics, but an area which has often required additional attention has been ESM, including TypeScript module resolution:

- "module": "node16" error: `This expression is not callable` in `postcss/postcss`
- disposable-email-domains: Use CommonJS export for "module": "node16" in `DefinitelyTyped/DefinitelyTyped`
- Module not found: Fully Specified ESM Imports (with `.js` extension) in TypeScript in `vercel/next.js`

While we have not yet invested the time to become deeply involved in shaping standards by activities like writing spec docs or becoming a TC39 champion, we have at times added feedback in existing discussions or contributed short proposal notes:

- Skip parameters in function parameter lists on Bluesky
- `await fetch.json(url)` proposal on Twitter
- Standard wire data format + form field error message UI for showing server validation errors without JS on Twitter
- Add style ordinal/cardinal to NumberFormat (RBNF) in `tc39/ecma402`

Tips for Contributors

I can highly recommend getting involved in Open Source - benefits include the ability to:

- learn about new technologies
- learn how to communicate and work with others
- network with developers in the community
- become familiar with interesting projects

There are plenty of resources on how to get started with open source, so I won't write another guide on that. If you're looking for a good place to start, try How to Contribute to Open Source by Open Source Guides.

Here are my more personal field notes for contributors:

1. Superpower: match the style and philosophy of the project
    - read the code of the project and try to match the style
    - also match the philosophy or goal of the project, if there is one
    - avoids wasting time on back and forth
2. Superpower: review your own contributions
    - read your own code as if you're the reviewer
    - try to find issues and fix them before submitting
    - comment on surprising or unusual parts of your contribution
3. Start small, but contribute widely
    - to ease into Open Source, make small contributions - small contributions are also helpful for others
    - if you resolve or work around an issue others have reported, add your approach if it's not already there
    - if you find a small issue, report or fix it
    - get into the habit of looking through the issues and pull requests of projects you use - soon you'll be contributing to a wide range of projects
4. Don't fall in love with your solution
    - be open to feedback and changes
    - acknowledge that your solution might not be the best one, or may not be accepted at all
5. Use AI carefully in your contributions

- AI can help you understand the codebase and suggest changes that match the style
- review AI-generated code carefully - it can produce low-quality output aka "AI slop"
- using AI tools can make the difference between contributing and not contributing

6. Use the Refined GitHub browser extension to simplify the GitHub interface and add helpful features
7. Report issues with enough information to make them actionable

   - explain what you were trying to do
   - describe what you observed
   - describe what you expected
   - include a minimal reproduction repo or code block
   - include the steps to reproduce the issue
   - include relevant version numbers
   - prefer code blocks over screenshots
   - think through the problem and provide a guess of what the problem might be near the end of the issue

Suggestions for Maintainers

During my time contributing to Open Source, I have also developed opinions on how projects can optimize their CX (Contributor Experience) for new contributors:

1. Optimize for contributions from web clients such as the GitHub web interface

   - avoid requiring a full local dev environment or terminal execution, also for contributions changing tests

2. Optimize for AI-assisted contributions

   - describe the project's style and provide instructions for LLMs in standard locations like `.github/instructions/*.instructions.md` or `.cursor/rules/*.mdc`

3. Simplify documentation and make it easier to understand for a wide audience

   - avoid jargon
   - prioritize clarity over purity / brevity - short explanations are often not enough for a wide audience
   - don't avoid repetition - it can help with navigation and comprehension
   - embed runnable examples and playgrounds in the docs

4. Provide a bug reproduction template

- example: Next.js repro template
- example: Reproduction Template of ESLint Stylistic
- example: GitHub template for creating a Rspack minimal reproducible example
- more examples: Awesome Open Source Automation

@karmatosed – Tammie Lister

github.com/karmatosed
maintaine.rs/karmatosed

Like many others, I discovered Open Source by necessity. I needed a system and software to run some projects, but couldn't afford one, so I turned to Linux. When starting a blog, I faced challenges with my system, but there was an open solution to help me. This experience has shaped my work, and it feels natural to build my career in Open Source, as it has given me so much.

For me, Open Source is fundamentally why I can pursue my career and earn a living. This simple fact makes it incredibly valuable and a core principle for me. However, it's more than just values, governance, and models. At its essence, the idea of giving back as much as you take is what resonates deeply. Open Source has provided me with so much; I've built my career around it. At this point, it feels as essential as air, and I strive not to take it for granted.

How do you grow your community?

Have a welcome

If you think of the project as your house, welcoming someone in is great. Even better is the tradition of a welcome mat and then sitting someone down, making them feel at home and give them a cup of tea. Your project version of that is solid documentation (no matter where it's hosted). This sets the tone of your project, the voice many will hear before they even start contributing to it.

Provide signposts to contribution

If you want contributions, be sure to show how and where they are wanted. It's one thing to say 'all contributions are welcome', but that's open-source theatre because honestly, it's likely not all are in every area as welcome as others. There are always more areas needed than some. Highlight these and make sure someone gets off to contribution success from the start.

Document everything possible

I've hinted at this with my previous points around a welcome and signpost, but you also need to have documented everything possible. From code to vision and everything possible – ensure there is something written about it if possible. You might want to have documentation as a

contribution, but if that contributor has to guess what you were thinking, that will not be someone who will stay around a long time. Give everyone a start with some basic documentation.

Show the small ways

By highlighting not just the big things to contribute to, but the little wins – someone can dip in and support right away. Show how to test, report a bug (using templates to make easier reporting is fantastic) and show the level of commitment of fixes. If something requires a certain skill or understanding, clearly mark that also – it helps people focus where they are going to be the most effective.

Say thank you

The biggest thing you can do to someone contributing is to say thank you no matter what size that contribution is. Recognise that contribution, say thank you. If someone takes the time to contribute, make sure you engage with them. You wanted them to, so don't leave them contributing without interacting. That's a lost contributor and one that will rapidly find another project.

Include and recognise contributions

Opening your project to contributions isn't a checkbox. It's not something to do if you genuinely don't want to accept and merge in contributions. Otherwise, it's performance, not actual, and you are unfair to people who try and contribute.

Beyond including, recognise those contributions. If you have a release, put those contributors are in front of it, celebrate them. They have helped you make what it is; you wouldn't be there without them.

Be open about roadmaps and releases

If you have contributors pretty soon, you need to be open about plans you have for your project, goals and releases. I believe you should have at least a project roadmap table somewhere—ideally, open project boards and a roadmap. I like the way Github does theirs.

Opening your project up can be incredibly rewarding, but do it respectfully and then who knows what incredible contributors will join you on the adventure.

What's the impact of AI on Open Source development?

The full impact is still being assessed. In the short term, this will simplify manual processes, including triage, reviews, and testing. This will enable work to occur in other areas of the project, allowing you to redistribute finite resources and have a positive impact quickly. It also means that the systems must earn trust.

As far as triage and review goes, we will be far from achieving full automation of this for quite some time. My instinct is that this is more of a help in clearing a percentage and making it easier for those working on issues. It can do that in significant numbers, to the extent that over 50% of the time spent in triage could be recovered I'd suggest sooner over later. I do see this going up over time, but how much depends on the quality of the modelling and our trust as a project.

With regards to testing, we have, over the years, even before the recent AI tech wave, been leaning into less manual testing. Issues and tickets needed less manual testing. We require testing for regression across all areas in our project. That's not AI, but we can take it even further, adopting a more updated approach to our testing.

Projects should focus on establishing a solid foundation rather than rushing to implement an AI solution while the field's landscape is still evolving rapidly. If they don't, the impact will be distractions and wasted time. If they do that, it will mean they are prepared. The most significant effect is that projects need to plan and not continue doing the same things they have been doing for so long; they also need to review all areas of their function.

At this time, with our capabilities in automation and AI in a world of agentic flows, we can do better than relying solely on manual processes. We must do so in order not to be left behind. It is also not resourceful to use humans this way. We can use our scarce human resources more sensibly. Our time is finite, and this project requires us to accomplish many tasks. There will be a balance between what we automate and utilise as resources and what we provide to a system. That, though, is something we need to start working out, and we do that by realising we need to find a balance in how we do things and measure the costs of our processes today and in the future.

What advice would you give to current and new maintainers?

You should put your seatbelt on first. A project reflects the state of its maintainers and core. If you are thriving, then it will. Make sure to document all tasks, automate as many manual processes as possible, and utilise every last hour effectively. Take time to think, not be reactive, chasing fires in tickets all the time or be at the mercy of notifications. Embrace public roadmaps and the art of triage. Above all, be kind to yourself and remember you are human and need a life, not just at a computer, because Open Source needs more of that from maintainers.

Want to read more from me? Please visit binatethoughts.com

Want to sponsor me? You can go through GitHub or also get in touch.

There is always sponsorship, of course, that is volunteered, and I'll do as much as possible whilst still keeping things flowing – let's get contributing!

@kgodey – Kriti Godey

github.com/kgodey
maintaine.rs/kgodey

I'm Kriti Godey, project lead and one of the maintainers of Mathesar. I've been a software engineer for almost fifteen years, and have been working on Open Source projects full-time since 2018.

Like most people, I stumbled into Open Source by accident. At 19, I was a CS major looking for hands-on experience, so I applied for a campus web dev job. By some mix of luck and good timing, the job involved an Open Source codebase—a CMS built on Django—and I got a crash course in both how Open Source works and why it matters.

I didn't know the term "maintainer", and I had no business poking around Django internals yet, but even the documentation blew me away. People had written all this detail? Together? Just so others could use it? I didn't realize I was already on the path to becoming someone who voluntarily writes style guides for issue templates.

"RTFM?" to "My PR was merged!"

My early career was built on Django, and as I worked with more libraries, I started hitting edge case bugs, and developed a compulsive habit of going straight to the source code when docs fell short. I lurked on mailing lists, learned to navigate bug trackers and got told to RTFM (correctly) on IRC. At one point, I definitely had the Django REST Framework file structure memorized. I also thought I was clever for finding undocumented functions and using them in production. That went about how you'd expect.

My senior honors project (a recipe recommendation webapp, in theory, anyway) was the first thing I ever open-sourced. It didn't actually work, the code was *terrible*, and the README was a to-do list. I pushed it to GitHub anyway, because I liked the idea that someone might find it useful or maybe mildly interesting.

The first time I felt like an *actual* Open Source contributor was when I fixed a misused word in the docs of a library I was using. It annoyed me until I realized I could just… fix it. So I opened a PR. The maintainer merged it with no comment besides an emoji. I was completely

thrilled to have actually *collaborated*, and both mildly mortified and weirdly proud to be called a nerd while doing it.

I didn't think of myself as "in" Open Source—I was just working, learning, and building things. But it was always there in the background. I met maintainers at PyCon and DjangoCon (even Guido van Rossum, briefly!), watched *Revolution OS* and got a little starry-eyed, and fixed the occasional bug I ran into. Over time, I started noticing the infrastructure that turned code into a community, and enabled people from wildly different backgrounds to work together and build something useful.

"Wait, now *I'm* merging PRs?"

In 2018, I was lucky enough to turn my admiration for Open Source into something more hands-on. Creative Commons was looking for someone to lead their engineering team, and I applied. It felt like a long shot, but I didn't want to pass up the chance to work with an organization dedicated to making knowledge more open. The job wasn't focused on community-building, but when I saw how many internal tools and libraries Creative Commons had, I got a little overenthusiastic in the interview and pitched a bunch of ideas anyway. I got the job—and then I got to actually implement those ideas.

Over the next couple of years, the team and I cleaned up and published dozens of repos, launched new projects, and built a contributor ecosystem from scratch. We participated in programs like Google Summer of Code and Outreachy, wrote documentation, and set up an Open Source community portal. I learned firsthand that code was just the tip of the iceberg. It's the documentation, responsiveness, onboarding, visibility, community architecture—that's what makes Open Source projects successful.

I also represented Creative Commons at the Open Source Initiative, where I got to meet maintainers from all over and talk shop. At some point, it hit me: I kept referring to them as "other maintainers." Which meant I was one too.

"What do we name the repo?"

When Creative Commons restructured at the end of 2020, I started looking for a new home for some of the work we'd been doing—especially our flagship project, CC Search (now Openverse at Automattic). That process led to receiving a grant from the Center of Complex Interventions to start what eventually became Mathesar, an intuitive UI that makes Postgres databases easier for non-technical users to work with.

The lessons I learned at Creative Commons about community building helped us design Mathesar to be contributor-friendly from day one. We labeled issues that were accessible to beginners. We

structured our internal team process around reviewing PRs quickly. We published specs, meeting notes, and design discussions on a public wiki. Even our "internal" team chat runs in a public Matrix room. The goal was to make it easy to show up.

But starting a project from scratch is a different kind of maintenance. There's no existing culture to steward—you're creating one. The project didn't just need code. It needed a shared vocabulary, a contributor workflow, a feedback loop, an ideation process, a technical identity. And that had to be invented in parallel—in public, with contributors across the world. I made a lot of mistakes. By the time we launched our alpha version in 2023, it was clear that this was a much bigger project than I originally thought it was, and that it was time to start thinking about long-term stewardship.

"Nonprofits need a financial audit *every* year?!"

We set up Mathesar Foundation (with generous support from Reid Hoffman) and now I'm both the project lead of Mathesar and CEO of the foundation. Mathesar has a team of full-time maintainers, and I don't get to write much code anymore. I still review design and implementation specs, weigh in on architectural decisions, and at least glance at every single issue and PR on GitHub. But my day-to-day is now mostly product direction, fundraising, and long-term strategy—a different kind of maintenance.

Open Source is often built for developers by developers. Mathesar isn't. Our goal is to offer infrastructure for *end-users* that has the full power and interoperability of Postgres, but with the flexibility and intuitive UI of a tool like Airtable. We're also 100% Open Source (GPLv3 and no "open-core"), nonprofit, and the project is fully self-hostable. My job now isn't just to lead the project—it's to make the project sustainable over the long term *without compromising any of those parts*. If we pull it off, I hope it makes the same path easier for other projects.

Of course, it's still hard to keep my hands off the repo. When GitHub released issue types a few months ago, I got very excited and impulsively decided to "improve" the repo. I deleted our "bug" and "enhancement" labels and replaced them with issue types. I thought everyone would appreciate the upgrade. Instead, I broke everyone's GitHub CLI workflows, and we ended up having to undo it all.

Personal reflections

I love that Open Source is shared by default. No gatekeeping, no judgment. Just: we made this, here it is, maybe it'll help you. It gives people a focal point to build something good around. It lets ideas evolve without permission. And it gives people tools that they might never have had the resources to build on their own.

The strange part is that you rarely hear from the people you help. But sometimes you do—and it's the best thing ever. A user sharing how they've been looking for something *exactly* like Mathesar. A bug report that's clear, thoughtful, and actually reproducible. A contributor who shows up out of nowhere with a PR that nails a tricky feature. A random Reddit post appreciating a tiny UX detail we spent three days on.

I still can't believe I get to do this full time.

Maintainer Month topics

This year, Maintainer Month is highlighting project security and the implications of AI on Open Source, so I wanted to add a couple of thoughts:

- **On key security features in Mathesar**: Mathesar connects to production databases, so we can't afford to be lax on security. In our recent beta release, we switched Mathesar's access control system to use PostgreSQL roles and privileges. This massively improves security, since permissions are enforced at the database layer rather than the API layer. We also use Django's default user system for UI authentication, since Django's a well-established framework.
- **On the impact of AI on Open Source**: AI is a tool—its impact depends entirely on how it's used. I can see it making maintainers' lives harder—by making low-effort PRs easier to produce, and easier by helping with triage, speeding up repetitive coding tasks, or improving documentation quality. I'm especially interested in its potential to make code and documentation interactive—something contributors can query instead of just read.

If anything here resonated—or if you just want to talk, I'd love to hear from you.

You can find me on GitHub at @kgodey, LinkedIn, or reach me by email at kriti@mathesar.org.

@leandromoreira – Leandro Moreira

github.com/leandromoreira
maintaine.rs/leandromoreira

I first got drawn to Open Source mainly out of curiosity about how things work under the hood. I remember being fascinated by emulators—these amazing programs that create machines within machines. Being able to read the entire codebase, run it locally, make changes, and even contribute improvements became my natural learning path. It was like getting a backstage pass to see how the magic happens.

What's Open Source to you?

To me, Open Source represents our collective digital inheritance. I firmly believe that nearly everything we build digitally today is only possible because we're standing on the shoulders of giants who share their passion, code, and hard work. It's a beautiful cycle—we use and improve upon others' work, then pass those improvements forward. That collaborative spirit is really the soul of what Open Source means to me.

What projects are you involved in?

Digital Video Introduction has become one of my most recognized projects. It offers an approachable yet comprehensive guide to understanding digital video technology. I break down complex media concepts using clear explanations and visualizations. What makes me particularly proud is seeing it translated into multiple languages and watching developers worldwide contribute to our shared mission of demystifying video technology.

My FFmpeg Libav Tutorial provides hands-on guidance for developers working with this powerful multimedia program. It walks through fundamental processes like decoding, encoding, and manipulating media streams with practical code examples and step-by-step explanations—essentially bridging the gap between theory and real-world implementation.

With Linux Network Performance Parameters, I explore optimization strategies for the Linux networking stack. This resource helps sysadmins and developers understand kernel parameters and sysctl settings that impact network performance, backed by practical recommendations for various workloads and environments.

CDN Up and Running aims to demystify Content Delivery Networks by explaining the core concepts behind content distribution, caching strategies, and performance optimization, including practical examples for implementation.

I'm also behind Redlock-rb, which implements the Redlock distributed locking algorithm in Ruby. With over 35 million downloads, it provides a reliable way to acquire locks across multiple Redis instances.

How do you grow your community?

I believe community growth comes down to two principles: kindness and reciprocity. Creating a welcoming environment where newcomers feel confident to contribute is essential. Being kind creates that safe space. And giving back is what truly sustains and grows both communities and projects—it's about nurturing that cycle of contribution that makes Open Source so powerful.

What's the impact of AI on Open Source development?

Being completely transparent, I'm not an AI specialist—my perspective comes simply as a developer observing the community and using AI tools myself. I see AI as potentially empowering developers and accelerating improvements by leveraging our collective body of work. We can build better things faster, but there's also risk in blindly accepting code suggestions or implementations without proper understanding. We could end up with codebases written by machines that few humans truly comprehend, creating maintenance and security challenges down the road. It's a powerful tool that requires thoughtful application.

Main contact points

- https://github.com/leandromoreira/
- http://www.linkedin.com/in/leandromoreira
- https://leandromoreira.com/

@lrusso – Leonardo Javier Russo

github.com/lrusso
maintaine.rs/lrusso

My name is Leonardo Javier Russo, I'm a software developer from Argentina. I got involved with Open Source 10 years ago when I designed a free and Open Source App for the visually impaired that was compatible with smartphones and tablets. It was an interesting experience where users and developers helped me with translations of the App to different languages.

The meaning of Open Source

I believe that when software is developed to contribute to society, it's selfish not to share the source code. In my opinion, the true value of a project isn't determined solely by the number of people it has reached, but also by how willing the developer is to share their work with other developers, so that they can improve it or offer alternative versions. Free and open-source software is the greatest expression of that line of thought.

Current projects

Today I'm working on LlamaWebServer, a web server that allows you to run an AI model locally and has a UI similar to WhatsApp. Also I'm working on Emulatrix, a Web that allows you to play retro games on a Web browser and where you can upload and download your saved games. The website doesn't have any ads and the user must provide the ROM file of the game that he wants to play, so that prevents to receive any copyright claims. Usually, when one of my projects reaches a MVP (Minimum Viable Product) stage, I write a press release and send it to different tech portals and journalists. By doing this, most of my projects were published in newspapers in America and Europe and also on television, like CNN and BBC. Also, recently the GitHub team got interested in my profile and after an interview, they published an article about my career on their website (The Readme Project).

Challenges

Most of the time, I think the main challenge is not technical, but related to the resources that are available for you. Sometimes it is the budget that you can spend or where you can find the

right person that can help you with a new design (UI/UX related). You can have a good idea for a project, but if you cannot present your thoughts and intentions in a way that everyone will understand what you are offering and why it's useful, you won't get far. The budget for a project is always important, but contributors can support maintainers in a lot of different ways, like: providing translations to different languages, reporting bugs, suggesting features or to connect a maintainer with a journalist.

Regarding new features, I always try to double check all the possibilities. For example, recently I designed a free and Open Source scraping bot and I wanted to use this bot and to receive an email from it with results, but I didn't want to pay for a server (node.js/smtp) for sending just one email (along with the effort, money and security practices that it requires). So eventually I created a Google Apps Scripts that provides an API for sending me an email. The API key is stored as a Secret in my repository and the scraping bot runs on GitHub Actions, so the cost is literally $0 and it's running in a secure environment where the API key is not compromised.

However, security is always something that you need to be focused on. I noticed when working for some companies that, sometimes, credentials were hardcoded in the backend. Usually a module that was assigned to some specific team a few years ago, no one really checked how it was designed but it worked, so years passed, the team members were gone and then you have a module in the company with hardcoded credentials and no one is aware of that until you get there to fix or update something. Services like CodeQL are useful tools that could help prevent that kind of issue.

AI

Lastly, I should mention that an AI could provide you the code that you may need, but usually a Senior developer will have to adjust or improve the quality of that code to have a technical solution that meets a standard and that can be used in a project. For example, when using Three.js, there are too many variables involved and the AI result can be erratic because of that. Eventually it could provide a new approach that could be useful, but in the end, a human developer needs to be there to adjust the code and make it work.

Contact information

https://www.lrusso.com

@martincostello – Martin Costello

github.com/martincostello
maintaine.rs/martincostello

I'm Martin, I'm a software developer based in the UK, and I've been a contributor to Open Source projects since 2013.

Open Source software is something that I have a lot of passion for, but it's not something that I've actively sought out. Instead, it's something that's organically grown out of my time working in the software industry. Incrementally as I've run into challenges, had ideas, or even just wanted to peek under the covers of how something works, I've found myself contributing back to projects more and more.

It all started with this pull request: Glimpse/Glimpse#493.

I'd been playing around with the Glimpse Open Source project, and ran into some difficulty configuring things to work correctly with the project I was trying it out with. I took the opportunity to raise an issue asking the maintainers how to resolve my problem. Once I got the information I needed to unblock my progress, I suggested to the maintainers that the help messages in the project could be updated to include additional information to help others who might run into the same problem in the future. They agreed, so I submitted my first pull request on GitHub, and it was merged just two days later.

Looking back, this highlights some of the things that I think are great about Open Source software:

1. Anyone can propose a change to a project - you don't have to be a maintainer or have a support contract to be able to contribute to a project you're using.
2. You can solve your own problems - if you run into an edge case in some software where it might not otherwise be prioritised by the maintainers, you can give it your priority and make the change yourself.
3. You can make things better for those who follow you - if you run into a problem, there's a good chance that someone else will have the same experience at some point in the future. Sharing your solution for a difficulty back to the project means you can magnify the impact of your discovery and leave things better than you found them.
4. You don't just have to implement a big new feature to contribute - even a small change to some documentation can be invaluable to not only other users, but to the maintainers as

well. Documentation is often the most neglected part of a project compared to the code itself.

Fast forward to today, and I'm a maintainer of multiple Open Source projects, some of which I started myself, others I've inherited from collaborating with others. These projects include:

- Polly - a .NET resilience and transient fault handling library
- Swashbuckle.AspNetCore - OpenAPI tools for documenting APIs built with ASP.NET Core.
- HttpClient Interception - a library for intercepting and mocking HTTP requests for .NET applications
- xunit Logging - An logging library for xUnit.net to route application logs to the test output

I'm also a regular contributor to .NET, raising issues, improving documentation, fixing bugs, and (very) occasionally adding new features. I also help with issue triage for ASP.NET Core, routing issues to the right core team members where necessary, or leaning on my own experience and knowledge to answer questions and troubleshoot users' problems myself.

But it's not just C# and .NET that I contribute to. I'm always happy to try and help out with projects written in other languages if they're a tool that I use and there's a problem I want to help out with (if I can). I've dabbled with Ruby, Go and JavaScript too.

Open Source software is a great way to *be the change you want to see in the world*. If you find a bug in something, rather than sit back and wait for someone else to fix it, you can take control of your own destiny and try to fix it yourself. Not only will you learn something new, you'll help others, and contribute back to the community that you're part of from consuming Open Source software in your own projects.

Over the last few years as I've got involved in more projects, especially as a maintainer of projects I've inherited, there's a few things I've observed that seem to be common pain points for maintainers of Open Source projects that aren't backed by a company or organisation. One of the major topics that should be a concern for consumers of Open Source software is that of burnout.

Many Open Source projects are ultimately run by a single person, and maintained in their spare time. When the popularity of a project grows past a certain point, it can become overwhelming for the maintainer to keep up with the volume of issues and pull requests to their project to keep the project healthy and their users' concerns addressed. In some cases this can lead to maintainers either being burned out by their experience, or having to abandon the project altogether due to a lack of time to focus on it amongst the other commitments in their life.

Abandoned projects can then become a security risk to the users of the software - the maintainer may no longer be available to fix (or accept fixes) a security vulnerability, or to publish a new

version containing a patch. This can lead to consuming projects with either an unpatched security vulnerability in their application, or having to expend time and effort to find a suitable replacement for the dependency and migrate their projects to use the alternative.

If you're a consumer of Open Source software, then you should consider how you can help contribute back to the projects you depend on to help keep them, and the wider ecosystem healthy. After all, the health of the projects you depend on directly impacts the health of your own projects too.

1. Find a bug? Raise an issue. If you find an issue, raise an issue for visibility and attract help for a fix. But take a moment to check whether there's already an issue for your problem (open or closed).
2. If you've found a bug, consider whether you can try and fix it too. Maintainers love pull requests to solve bugs in their projects, as it often makes resolution quicker, and also helps avoid considerations over prioritisation of the issue compared to other issues in their backlog. Just be sure to check the contribution guidelines for the project first.
3. No contribution is too small. You don't just need to submit a cool new feature or fix a bug to contribute to an Open Source project. Documentation is often an overlooked part of a project, but it's just as important as the code itself to help users succeed. If you find a typo, or something that's unclear, consider raising an issue or submitting a pull request to fix it. Using the GitHub web interface is a great way to get started with small changes. You could be done within just a few minutes.
4. Sponsor a project. An increasing number of Open Source projects now accept donations via GitHub Sponsors, and these don't have to be a large amount of money, or even an ongoing commitment. If you get value from a project and are in a position to do so (especially if you're using it in a commercial project), consider sponsoring the project to help the maintainer prioritise the maintenance of the project amongst their other responsibilities and commitments. Even a small amount as a one-off sponsorship can go a long way to helping the maintainer feel appreciated for their work.

Open Source software is ultimately a large collaborative effort, with projects depending on each other to solve problems in the best way they can to help users reach their goals, whether that's in industry, academia, charity, or just for fun.

I hope this post has inspired you to consider how you can contribute back to the Open Source projects you depend on.

@mikemcquaid – Mike McQuaid

github.com/mikemcquaid
maintaine.rs/mikemcquaid

Hi, I'm Mike McQuaid, the Project Leader for Homebrew and CTPO of Workbrew.

I've worked on Open Source software in some form for 20 years. I got started in a fairly traditional way, using desktop Linux in university in the early 2000s. Through this I ended up helping people out in IRC channels, submitting bugs and then patches on bug trackers and modifying existing software for my own use.

After a previous failed attempt, my big jump into Open Source came through my work on Google Summer of Code for KDE in 2007. I stayed involved after this as a KDE contributor for a few years until I moved primarily to macOS. Shortly after this in 2009, I started contributing to and then maintaining Homebrew, the macOS (and now Linux) Open Source package manager. My proudest non-Homebrew work is a single, random commit to the Linux kernel under my old name ("Mike Arthur") in 2006. Most of my Open Source work has been in the Homebrew ecosystem for the last 10 years but I've dabbled with various other projects.

Homebrew

I was the third maintainer involved in Homebrew, after Max (the creator) and Adam (who I joined). Since Max stepped down, I've been in an informal leadership role and as the elected "Project Leader" since 2019.

I've spent a lot of work on Homebrew not not just the engineering aspects but also trying to make the project more sustainable. This has included:

- working on a "contributor funnel" from user to contributor to maintainer
- figuring out how best to ask people for money to support the project
- fundraising for our initial CI hardware and building partners with companies like MacStadium so we can afford a better CI system
- general fundraising and finding a fiscal host for the project (Open Source Collective, previous Open Collective Foundation and Software Freedom Conservancy)
- creating, encouraging and improving automation to scale ~30 maintainers to millions of users

- building a community of maintainers, many of whom meet up in person once a year at Homebrew's AGM
- encouraging a culture of maintainer support, boundaries and intolerance to bad community behaviour
- defining and reducing the support footprint so we're able to handle millions of users

Challenges

The hardest part of all this has been (and likely will continue to be) helping maintainers (including myself) to not burn out while keeping users happy. Too many Open Source users are overly entitled and think a bad bug is an excuse to be rude or demand a quick fix. Sometimes, things can get darker still with things bleeding into personal abuse and harassment. I once had someone say they were going to turn up and harass me at a conference talk I was giving. I'm lucky enough to not be threatened by stuff like this but: it's completely unacceptable behaviour in our community and we should be more aggressive in shutting it down.

Contributors

Contributors have been a huge part of Homebrew's success. Our contributor to maintainer ratio is >100:1. We've achieved some of this with automation and some of it just trying to make it as easy as possible to contribute and get feedback without needing a human to help. Most of our contributors are great. If you want to be a great contributor, please:

- don't argue with maintainers, assume they know better about the software they are maintaining (until you're also a maintainer)
- try to address your code review comments in a timely manner
- don't open PRs you aren't willing to finish
- don't expect people to tell you exactly how to implement something; there's a point at which it'll just be quicker for the maintainer to do it
- feel free to use AI tools but ensure you're very carefully reviewing the output yourself and not deferring that to maintainers

Security

As a package manager used by millions, Homebrew is very conscious of our security profile. We continue to try and improve it over time but some of the best practises we've introduced have been:

- being a maintainer is a responsibility, not a privilege, and if you're no longer doing maintainer work: you will lose your access
- we avoid giving everyone access to everything but instead give people as little access as they can to do their work
- we lean heavily on automation but ensure we always have a human verifying/confirming/approving results before it is shipped to users
- we use GitHub functionality for many things and try to integrate as many of their security features as possible
- we try to design each component of Homebrew to not fully trust any other component or piece of infrastructure
- we encourage users to adopt secure configurations (e.g. macOS versions that receive security updates)
- we have had several third-party penetration tests

The biggest challenge facing Open Source security today is filtering out the signal from the noise. We get large amounts of "drive-by security spam" where people have claimed to find some vulnerability with our website that's e.g. statically generated. Similarly, we end up with a lot of reports from novices with a passion for security who can be overly fixated on threat models that professional security researchers consider insignificant. All of this takes a lot of time and energy away from legitimate security problems and means security disclosures are, sadly but correctly, assumed to be non-applicable more often than not.

AI

LLM AI tooling is becoming increasingly widely used in our industry. I use it fairly heavily myself; mostly as a good autocomplete (even when writing this article) rather than generating entire files/posts/PRs.

I think LLM AI tooling has the potential to be positive for Open Source but the jury is still out for now. To be really effective: you need to ensure you do extensive review of any AI generated output. Who are some of the best people in the world at extensive code review? Open Source maintainers. This is why I think it could be useful for those folks.

Sadly, some contributors have used it to generate entire PRs they don't understand and expect the maintainer to review their AI slop for them. This slows everyone down and, much like the security spam above, poisons the well for everyone.

Advice

I've worked on Homebrew for 16 years this year and never taken more than a couple of weeks "off" in that time. I enjoy it and seem to be resistant to the burnout that's affected other maintainers.

The main reason I think I've been able to do this is in my mantra (and blog post): "Open Source Maintainers Owe You Nothing".

I encourage every maintainer to read and internalise this. Unless it's your full-time job to work on your open-source project: you can walk away at any time, guilt-free. Additionally, no-one can make you do anything you don't want to do.

When I was employed at GitHub I (pretty much single handedly) built the "archive a repository" feature exactly for this reason. I wanted to allow people to walk away and not get notifications any more while allowing others to view and fork their work.

Once you start to dread getting issues or contributions on your Open Source project: it's probably time to leave.

If it's not time to leave: think about the parts of it that fill you with dread and consider how you can adjust your documentation, policies, templates, code or even just personal boundaries to not have to do these any more.

Most of all though: good luck. It's not always easy. If it was: everyone would be doing it. You're contributing to the biggest collaborative effort humankind has ever tried. You are a hero for even trying. Keep up the good work!

@mte90 – Daniele Scasciafratte

github.com/mte90
maintaine.rs/mte90

My name is Daniele Scasciafratte, and I started contributing to Open Source projects over ten years ago. My journey began in 2013 with Mozilla and in 2015 with WordPress, where I took on various roles and participated in different activities.

At Mozilla as an example, I was part of the Mozilla Reps council for two years and contributed to numerous projects, from community management to development. My name is listed in the about:credits section of Firefox and on the monuments outside their San Francisco office.

For WordPress instead, I was a core contributor for years, co-organizer of various meetups, speaker, and localizer.

From december 2020 to april 2025 I had an Italian weekly podcast about Open Source and technology (200~ episodes in 4 years).

Now I am focusing on other projects and communities, internet is changed but also the open source world.

What's Open Source to you?

To me, it is a way to extend human knowledge and what the world can offer to the people. This means that you need to learn how to talk with people and relate with them to create a real Open Source project. The effect that open source can have to the world but at same time to your career is something difficult to see in the world of non-profit voluntary activities.

What projects are you involved in?

Currently, I am serving my second term on the council of the Italian Linux Society and am part of the maintainer group for the Amber language.

Additionally, I have written a free and Open Source book titled "Contribute to Open Source the Right Way," available here, and I am working on the fourth edition.

How do you grow your community?

Growing a community depends on the project's topic and the type of contributors you want to attract, whether they are developers, localizers, etc.

It's not easy, which is why I wrote the book. In my experience, the activities you offer to the community and the promotion/evangelism around them make the difference. So the first step is how we can transform the project to be more community friendly but also more easy for a new comer to contribute? In this way we improve also the brand awareness and the project quality in a single (not tiny) task.

What are the main challenges you face as a maintainer?

Keeping up with everything alone is not easy, and it's important to build a team to focus on different aspects with people you can trust.

Time management is crucial, as you need to handle many things alone or with others and learn how to "bother" people based on their time zones.

What are some ways contributors can better support maintainers?

In my book, I suggest that users and contributors take the first step to help a project by communicating with it. It's a learning curve that, once started, can only grow.

No one needs complains, everyone needs a feedback with suggestions about how to improve it.

What's the impact of AI on Open Source development?

AI can be a great support in Open Source development, but you can't blindly trust what it generates. It can help automate repetitive tasks, review code, and manage issues, but with code you can't thrust them.

Contact information

- https://github.com/mte90
- https://daniele.tech
- https://twitter.com/mte90net

@nickytonline – Nick Taylor

github.com/nickytonline
maintaine.rs/nickytonline

Hi, I'm Nick Taylor (@nickytonline), a Developer Advocate and software engineer based in Montreal, Canada. Over the past several years, Open Source has been the thread connecting every chapter of my career — from learning new technologies to building communities and landing roles at companies like Forem (the team behind DEV), Netlify, OpenSauced, and now Pomerium, where Open Source continues to be at the core of my work.

Getting Started with Open Source

Early in my career, all my work experience was in closed-source environments — and Git wasn't even part of the workflow yet. My journey into Open Source began out of a desire to learn Node.js and React, technologies I wasn't using in my day job at the time.

My first pull request wasn't perfect (far from it) — but that's the beauty of Open Source: learning by doing. I started contributing to projects like React Slingshot and eventually became a maintainer. That experience showed me that Open Source isn't just about code — it's about community, mentorship, and collaboration.

What Open Source Means to Me

Open Source is more than just putting code out into the world. It's about creating spaces where people can learn, share, and grow together. It's about giving back, fostering trust, and helping others on their own journeys.

Contributions can come in many forms — from opening issues to improving documentation to triaging bugs. **Every contribution counts, not just code.**

What I'm Working On

Right now, my Open Source focus is mostly on:

- **Pomerium**: I help build and advocate for Pomerium's open core Zero Trust access platform. That includes documentation, demos, and improving developer experience.

- **Copilot Extension Template**: I maintain this starter kit for building GitHub Copilot Extensions.
- **Fun Product Manager Copilot Extension**: I built this playful AI-driven project to show off the template's flexibility: Fun Product Manager Extension.

In the past, I've worked at **OpenSauced**, improving contributor onboarding, Open Source analytics, and their AI feature StarSearch — Copilot for Git history. At **Forem** (DEV), I contributed to the Open Source platform. On the **Frameworks team at Netlify**, I supported frameworks like Next.js, Remix, and Astro.

I also contribute to projects like Chatty by Addy Osmani and Unsight.dev by Daniel Roe.

Growing Communities (Not Just Chasing Stars)

For me, building a strong Open Source community means creating a welcoming environment. Some things that help:

- Clear contributing guides and documentation
- Labels like `good first issue` to support new contributors
- Patience and encouragement during code reviews
- Using conventional comments or similar approaches for constructive feedback

One thing I learned at OpenSauced is that stars alone don't define success. Forks, active discussions, and contributor engagement paint a much clearer picture. **Community > Clout.**

Challenges I Face as a Maintainer

One of the biggest challenges I deal with is balancing Open Source contributions with full-time work and life.

Automation, strong documentation, and setting healthy boundaries help — but building and sustaining a welcoming community takes constant attention and care.

How You Can Help as a Contributor

- Follow issue and pull request templates. They're not meant to annoy you — they save time and reduce back-and-forth.
- Ask questions when you're unsure — communication matters.
- Be patient — maintainers juggle a lot.

- **Remember that non-code contributions like documentation, triage, and community support matter too.**

Empathy goes a long way toward making Open Source projects sustainable.

How I Approach Open Source Security

Security is an ongoing responsibility. Some practices I follow:

- Keep dependencies updated
- Review third-party libraries carefully
- Use CODEOWNERS for consistent reviews
- Sign commits when possible

At OpenSauced, I helped bring security front and center by introducing Software Bill of Materials (SBOMs) (PR #3938) to make project dependencies more transparent. I also documented the approach on the OpenSauced blog (post written by my coworker Bekah).

The Biggest Security Challenges I See

Dependency sprawl — relying on countless libraries — is a major risk. Supply chain attacks are on the rise. Another serious challenge is maintainer burnout, especially as more security responsibilities fall on individuals without enough outside support. **Building a trusted team of co-maintainers can help spread the load and make projects more resilient.**

We need better tooling, more community investment, and shared responsibility to maintain a secure Open Source ecosystem.

Open Source as a Gift Economy

Chad Whitacre from Sentry said it best on a Pomerium live stream: Open Source is a **gift economy**.

When we contribute Open Source software, we offer a gift to the world — no strings attached.

But receiving that gift creates an **invitation** to give back, whether through code, support, sponsorship, or documentation.

This framing keeps Open Source grounded in generosity, while also acknowledging the need for intentional and thoughtful investment. Initiatives like the Open Source Pledge bring this spirit to life.

How AI Is Changing My Work in Open Source

AI is already reshaping how developers build and contribute. Tools like AI code completion accelerate development but raise new questions around authorship, licensing, and trust.

Another important shift is that developers often take on more of a **code reviewer** role when using AI coding assistants — evaluating, correcting, and adapting generated suggestions instead of writing every line themselves. This changes not just the pace of development but how responsibility and accountability are handled in Open Source contributions.

My Advice to Maintainers (New or Not)

- Be kind. Words matter.
- Document everything you can to make it easier for others to contribute.
- Automate repetitive tasks when possible.
- Celebrate contributors and milestones.
- Take care of yourself — sustainability is key.

Most importantly: **keep Open Source fun**.

Passion and community are what make this all worth it.

References and Projects

- GitHub Profile
- React Slingshot project
- OpenSauced blog: Growth Hacking Killed GitHub Stars
- OpenSauced blog: Security and SBOMs
- OpenSauced PR: Add SBOM support
- Copilot Extension Template Project
- Fun Product Manager Copilot Extension
- Chatty contribution PR
- Unsight.dev contribution PR #1
- Unsight.dev contribution PR #2
- Pomerium Live Stream: Funding in Open Source (with Chad Whitacre)

About the Author

Nick Taylor is a Developer Advocate at Pomerium, where he works on infrastructure, Zero Trust security, and cloud-native networking. A long-time Open Source contributor and community builder, he focuses on empowering developers through practical solutions, technical demos, and real-world education. Nick is passionate about making complex technologies more accessible and building spaces where developers can grow and succeed.

@niklasmerz – Niklas Merz

github.com/niklasmerz
maintaine.rs/niklasmerz

My name is Niklas Merz and I first got into Open Source to scratch my own itch. It started with tinkering on my own little projects and some small contributions to Open Source projects I used for my personal and professional work. Over the years, I shifted from contributing code to taking on some community work. I no longer use that project in my day job, but I still like to give something back and help it move forward.

Let's start at the beginning. Like many others I did smaller documentation and code fixes because I used Open Source projects for my work. At some point during my apprenticeship I got the opportunity to work on a mobile app built with web technologies and this led me down a path to joining a project management committee (PMC) of the Apache Software Foundation (ASF) project Cordova. The ASF has the official motto of "Community over Code" and after some years in this space I can relate to this more than ever.

Everybody finds a different entry, but I think the most sustainable way might be to get active in Open Source is from necessity. When I started using Apache Cordova, contributing became logical very soon. Projects like Cordova that have an open plugin ecosystem are naturally good for starting out. At some point in my day job we wanted additional features no existing plugin offered, and I started to build my own plugins and made them Open Source because it's just fair to give something back. We also encountered some issues or limitations in the core of the framework, so I dug deeper into the project and tried to fix them myself. From my experience it was just much faster to get help from the community if you can show that you did some research and offered solutions to tackle the problem. Every Open Source community is different, but usually it's appreciated if you show as much initiative as you can.

My Open Source journey started out with coding fixes and features but over the years became much more. I'm proud of the features I implemented, the releases I managed, and the contributors I supported. Lately I really enjoy sharing my expertise as Co-Chair of the W3C WebView Community Group to improve the core technology that made Cordova possible. I also became interested in the community side and attended events to meet people and give talks. You can learn a lot from the wonderful people in Open Source and form meaningful connections. My current job came from a connection that formed at an Open Source event. Along the way, I also began to notice just how different Open Source projects can be in how they are built and

maintained.

Types of projects

After some time in Open Source you find out there are very different types of projects and maintainers with different ways of doing Open Source. If you are planning on using or contributing to a project I think it's important to check if the project is healthy and active, but also understand how it's run.

An Open Source project can be just a single person working on their project in public and accepting contributions from others. That's where I started. I built a plugin that was useful for me and published it on GitHub under the MIT license, so everybody can use and improve it. Over the years this plugin got many releases, issues and contributions. Some parts of the plugin were even rewritten entirely from other contributors and I just did code reviews, testing and publishing. The experience you get from managing your own project prepares you with the basic skills of tools like Git, GitHub, CI and security best practices but most importantly communicating with strangers from all over the world. As a maintainer good communication and patience are very important because that helps new people join and stay in your community.

Another common type of Open Source project is the one stewarded by larger Open Source foundations like the ASF, Eclipse, Linux Foundation etc. These projects usually have a bigger number of maintainers. The foundation's job usually is to provide oversight and rules for important things like license and security compliance, release process, trademarks, funding and much more. If you are looking into contributing to these foundation hosted projects you may be required to make some extra steps and follow additional processes. Most projects try to make contributions as easy as possible and provide useful resources and documentation. No matter the structure, projects rely on people with different strengths. Over time, I started to see recurring patterns in the kinds of roles maintainers play.

Maintainer personas

There are different types of people typically needed to run a bigger project successfully and sustainably. For a talk at the ASF event "Community over Code" I tried to define them and give them names:

- Silent Hero
  - Prefers to stay in background
  - Does a lot of coding work
  - Handles many important and sometimes boring tasks like releases

  - Is a driver of progress

- Helping hand
  - Acts as the contact person for the community
  - Answers a lot of user questions
  - Takes care of issue triage and response

- Advocate
  - Often is seen as the voice of the project
  - Gives talks and takes care of promotion
  - Works on websites, documentation, podcasts etc.

- Founder/Legend
  - Started the project or is a contributor since "the early days"
  - Has lots of deep knowledge
  - Knows the history behind things
  - Sometimes is hard to reach because they may have moved on to other things

As you can see there are many roles in Open Source you can fill and all are important for a project to succeed. It's not just about coding as many people assume, but there are lots of more important tasks that make the code usable by consumers. If you're part of an Open Source community, you can use these maintainer personas to critically reflect on your own and your fellow maintainers' roles. I'm sure the list is far from complete. Understanding these roles helped me reflect on how to check how the project works and if the project is healthy. That's something every maintainer should think about from time to time.

Conclusion

Open Source has taken me from fixing small bugs to building communities and sharing knowledge. It's not just about code, it's about people, collaboration, and making things better together.

Advice for maintainers

Every maintainer faces challenges like burnout, lack of funding or missing new contributors. For me personally motivation to continue working on some projects decreased, because I no longer really use them daily. Nevertheless, I'd like to stick around and try to give back by doing community work like organizing meetups or helping out on releases. My advice to maintainers of projects that are larger or older is to do health checks from time to time. For Cordova I wanted

to check if our perception matches the reality of the project and check on the user base. We ran a survey to find out who is using Cordova, which parts of the framework are most used and which possible pain points exist. Taking a step back to evaluate the size and health of your project and community can help you refocus and rediscover your motivation.

Advice for contributors

Everybody's experience is different. It's important to start somewhere and to stick around to grow. You need to find projects that you're motivated to work on and just get started doing the work that fits your skills and interests. If you do your best and show initiative, maintainers will appreciate your work, and it will make a difference. It's really rewarding to see that others are benefiting from your contributions. Be patient and open-minded. You'll never know where this journey will take you and which people you'll meet along the way.

Contact

https://github.com/NiklasMerz

https://www.linkedin.com/in/niklas-merz/

@nolanlawson – Nolan Lawson

github.com/nolanlawson
maintaine.rs/nolanlawson

I first got involved in Open Source in the early 2010's. Back then, it was a fun hobby project – a good excuse to flex my coding muscle, gain a little clout, and put my vision of software out there in the world.

Nowadays, Open Source is still the realm of tinkerers, dreamers, and hobbyists, but it has also exploded in popularity. Nearly every piece of software in the world relies on it. And yet, a lot of our practices around Open Source have not evolved to meet the challenges of this new reality: supply chain attacks, maintainer burnout, and never-ending dependency upgrades are all problems that we've only begun to grapple with.

In this article, I'd like to humbly offer some advice to my fellow maintainers to meet the moment of today's Open Source landscape.

My background

My first major Open Source project was PouchDB, a JavaScript database. I got involved because I felt I could offer something useful to the project – bug fixes, performance improvements, new APIs. And plus, it was fun.

However, I quickly learned that spending every weeknight and weekend toiling away at open GitHub issues is a quick path to burnout. I wrote an article on the topic that struck a nerve with the Open Source community, and eventually coincided with my stepping away from the project in 2017.

Since then, I've learned a lot about how to pace yourself with Open Source projects, how to avoid burning out, and when to call it quits. My key advice to new maintainers would be:

You do not owe anyone anything. There is a reason Open Source licenses have all-caps warnings like "THE SOFTWARE IS PROVIDED 'AS IS,' WITHOUT WARRANTY OF ANY KIND." There can be a perverse effect in Open Source where the more work you do, the more is asked of you. Remember that just because you did work in the past does not give others a right to demand more from you, no matter how urgent they may sound in a GitHub issue.

If it's not fun anymore, you can step away. Think about why you got into Open Source in the first place. Was it to scratch an itch? Explore a new language or framework? If the conditions that led you to Open Source in the first place have changed, then you should reflect on whether it makes sense to keep doing it, or whether you should move on to other projects.

Leaving a project is not the end of the world. As my friend Jan Lehnardt has said, "The tech comes and goes, but the people remain." You can step away from a project without sacrificing the community or reputation you've built up in the interim. And in the end, your software still *works* and delivers value for people, even without you actively working on it. And by leaving, you may even open up space for newer contributors to fill your shoes.

Making Open Source work for the long haul

Despite my experience with burnout, I do still believe that Open Source is a force for good in the world, and that there is value in being a long-term maintainer of a project. But I've also learned some techniques that can help reduce the maintenance burden and keep burnout at bay.

I still maintain several small Open Source projects, but my years of experience have caused me to temper the naïve enthusiasm of my early days. Here is the advice I would give to maintainers who want their projects to thrive in the long run:

Be careful adding new features. In any successful Open Source project, the number of feature requests tends to grow (perhaps until it can read email). But due to the Pareto Principle, *these are not the same requests*. You may find that you have 10 people asking for 10 different features, but each one is only useful to those 10 individual people. And if you add all those features to the project, you make your project more complex, harder to maintain, and harder for new users to understand. Sometimes it is worth saying "no" to a feature request, or suggesting that the user do what's great about Open Source: just fork it!

Reduce dependencies. This is not only helpful for long-term maintenance, but also supply-chain security. Every dependency is something that must be tended to and upgraded, especially as the laundry list of CVEs piles up. Do you really need to support Internet Explorer 9 and Node.js 6? Do you really need a utility library to pad a string? If not, unburden yourself and trim your project down to the bare essentials.

Avoid breaking changes. Many Open Source maintainers have taken Semantic Versioning as an invitation to make breaking changes whenever they feel like it. Don't like the way you named an API? Just rename it! However, think about the maintenance burden this places on consumers of your software (who may be other Open Source maintainers!). Every breaking change is something you have to document (potentially forever), and it immediately obsoletes

the informal documentation built up in Stack Overflow, blog posts, etc. Breaking changes should be as infrequent and minimal as possible.

Conclusion

I love Open Source, and I'm grateful for the friends and experiences I've gained from it, as well as the talented pool of other Open Source maintainers who keep the whole system humming. I do believe, though, that we maintainers should take some responsibility for the software we've built, and to be cognizant of the way the rest of the world has grown to rely on it.

Of course, the burden does not fall on us alone. The "random person in Nebraska" is not to blame for the rest of the world relying on their work: as I said before, they don't owe anyone anything. Companies that use Open Source should make an effort to give back – either through money or time – to the software that sustains them.

That said, we don't live in a perfect world, and currently Open Source is in a strange moment where it's often created by part-time hobbyists for the benefit of stupendously profitable corporations that give very little back. There is still value in doing it, and the relationship can be reciprocal, but maintainers often have to navigate this world alone, and to jealously guard their own mental health and the long-term health of their projects.

If you are just now getting into Open Source, I would say: wonderful! It's desperately needed, and you will likely learn and grow in a million ways throughout the process. Do try, though, to be respectful of your own well-being, and of the well-being of a world that has taken Open Source as a critical dependency. Very likely, the world will be counting on the next generation of maintainers to do it well and to do it sustainably.

@notmyfault – Alexander Brandes

github.com/notmyfault
maintaine.rs/notmyfault

My journey into Open Source started back in 2013. I was heavily into Minecraft at the time, and with a group of friends, we started hosting our server. That quickly grew into building custom plugins, which meant I had to teach myself Java. Over time, what started as a fun project turned into something serious.

As I built more, I started looking at existing Open Source Minecraft plugins on GitHub—using them, learning from them, and eventually contributing back with bug fixes, feature improvements, and optimizations. That was my first real exposure to contributing to an Open Source ecosystem. It taught me not only how to write code but how to collaborate with other developers, navigate pull requests, and maintain community-driven software. Those experiences laid the groundwork for everything I do in Open Source today.

What's Open Source to you?

Open Source is about empowerment, collaboration, and shared learning. It's a space where anyone can get involved and make a difference, regardless of background or credentials. It's also where I've learned the most—both technically and through working with people from all over the world.

What projects are you involved in?

These days, I'm primarily involved in the Jenkins project. I contribute to Jenkins Core, maintain several plugins, and help with weekly and Long Term Support (LTS) releases. I also serve on the Jenkins governance board, where I work on community processes, release coordination, and broader project direction.

Beyond development, I focus on documentation, infrastructure improvements, and helping new contributors get involved. Jenkins is a large and diverse ecosystem, so there's always something meaningful to work on.

How do you grow your community?

The most important part of growing a community is making it welcoming. I try to support contributors by reviewing pull requests, answering questions, and mentoring wherever I can. It's significant that people feel like their time and effort are valued.

We also put great effort into making our processes transparent—clear contributing guidelines, labels for good-first-issues, and open communication channels all help. When contributors see that their work has real impact, they're more likely to stay and grow into leadership roles themselves.

What are the main challenges you face as a maintainer?

Time management is probably the biggest challenge. Between reviewing code, fixing bugs, releasing updates, and supporting users, there's always more to do than hours in the day. Many maintainers—including myself—contribute in their free time, which adds to the balancing act.

There's also the challenge of scaling knowledge and processes as the project grows. Making sure contributors understand best practices, architecture, and long-term goals takes ongoing effort.

How can contributors better support maintainers?

Small contributions really do go a long way. Clear, well-documented pull requests make a huge difference. Testing changes before submitting them, writing or updating documentation, helping triage issues—these are all incredibly valuable.

It also helps when contributors stay engaged. Taking ownership of a plugin or feature, following through on feedback, or helping others in the community can lift a lot of weight off maintainers' shoulders. Open Source is a team effort, and shared responsibility makes it sustainable.

What are some of the key security practices you've implemented in your project?

Security is a core part of maintaining software responsibly. In my work, I focus on minimizing dependencies, using secure defaults, reviewing code carefully, and staying up to date with patches.

Jenkins also has a dedicated Security Team that handles vulnerability reports and coordinates responsible disclosures. Their work is essential, but every maintainer has a role to play in keeping things secure—especially when touching sensitive parts of the codebase or publishing new releases.

What do you think are the biggest security challenges facing Open Source today?

The scale of modern Open Source use is both a strength and a challenge. A small project maintained by just a few volunteers might be used in thousands of production environments. That creates pressure to maintain a high level of security with limited resources.

Dependency management is another major issue. A vulnerability in one upstream library can have a ripple effect across hundreds or thousands of downstream projects. We need better tooling, processes, and collaboration across ecosystems to manage that complexity.

What advice would you give to current and new maintainers?

Start small, stay curious, and ask questions. You don't have to know everything to be a good maintainer—what matters is consistency, communication, and care for the project and its people.

Focus on building relationships, not just writing code. Good documentation, mentoring others, and being open to feedback all help build a strong, resilient community. Open Source is a long-term effort, and it thrives when we support each other.

@patrickheneise – Patrick Heneise

github.com/patrickheneise
maintaine.rs/patrickheneise

I work full-time building solar trackers at Nevados Engineering, but my path through software development has been paved with Open Source every step of the way.

The Beginning

In the mid-90s, I crafted my first website when the internet was still in its infancy. It was 1996, and the web was a very different place - table layouts, blinking text, and "under construction" GIFs were everywhere. Little did I know that this would be the start of a decades-long journey in software development. I was captivated by the ability to create something and instantly share it with the world.

After studying Computer Science in Media in Germany and Media Technology in the Netherlands, I found myself increasingly dependent on libraries and tools that were "just there" - Open Source software that I used long before I even understood what Open Source really meant. Java was my gateway into this world around 2001, introducing me to a vast ecosystem of freely available code.

The more I learned, the more I realized how much I owed to this invisible community of developers who had shared their work. Every project I completed relied on foundations built by others who had chosen to give their work away freely. This realization gradually shifted my perspective from being just a consumer to wanting to contribute something back.

Finding My Place

I wouldn't consider myself a maintainer - not even close. When I think of maintainers, I picture heroes like Matteo Collina maintaining the entire Fastify ecosystem, Sindre Sorhus with over a thousand little helpers on "npm", Daniel Stenberg with curl or the countless others who dedicate themselves to keeping the digital infrastructure of our world running. My relationship with Open Source has been different, but no less meaningful to me.

I've found my place in Open Source by building communities. In 2012, after visiting BerlinJS and experiencing the energy of their meetups, I was inspired to start BarcelonaJS. The model

was simple: create a space where developers could come together, share knowledge, and build connections.

Those early meetups were small but passionate. We'd gather in various spaces around Barcelona - sometimes offices, sometimes bars - and share what we were working on. I remember one night when a developer showed a project they'd been struggling with for weeks, and within minutes, three others had jumped in with suggestions. By the end of the evening, the bug was fixed, and new friendships were formed. The community grew organically, fueled by a shared enthusiasm for JavaScript and web technologies. Before long, we were hosting regular events with dozens of attendees and even launched NodeConf Barcelona and MediterranéaJS.

When my wife and I adopted a location-independent lifestyle in 2017, I founded Zentered.co, a software engineering and consulting firm, as an Estonian e-Resident. As we traveled through Europe and Southeast Asia, I carried the community-building ethos with me. When life took us to Cyprus, I continued this mission by starting CyprusJS and the Cyprus Developer Community (cdc.cy). Now in Boulder, Colorado, I've restarted BoulderJS with the same vision.

For me, organizing meetups and conferences around JavaScript and Open Source has been my biggest contribution to the ecosystem. I love bringing people together, watching them learn from each other, and seeing the excitement when someone builds something new or solves a challenging problem. These communities have become incubators for collaboration, friendship, and innovation.

The Projects

Unlike the hundred lines of code that eventually became curl, my technical contributions to Open Source have been more modest but still meaningful to me. I've created a few pet projects and libraries - GitHub Actions and workflows for managing real-world events and meetups (called GitEvents), and developer tooling for Vercel, GitHub, and Cloudflare.

GitEvents emerged from a practical need to streamline the organization of tech meetups. Having run communities in multiple countries, I recognized patterns in how events were managed and saw an opportunity to automate many of the repetitive tasks. The project uses GitHub Actions to handle everything from speaker submissions to event announcements, making it easier for anyone to start and sustain a tech community.

My developer tooling projects were born from my own workflow frustrations. Each time I encountered a process that felt inefficient or repetitive, I'd create a small utility to address it. Over time, these utilities evolved into more substantial projects that others found useful too. The feedback loop of releasing something small, getting user suggestions, and iterating is incredibly rewarding.

I don't think of these as projects I "maintain" - they're more like gardens I tend to when I have time, with no pressure or obligation. If someone finds them useful, that's a wonderful bonus.

The Challenges

Time is always the biggest constraint in my Open Source contributions. With a full-time job at Nevados Engineering, where we're building innovative solar trackers that don't require grading, and family commitments, I can only dedicate a few hours a month to my Open Source projects. I have a backlog of ideas and improvements I want to implement, but I can only focus on a few at a time.

When users open issues or request features for my projects, I try to encourage them to submit PRs rather than just reporting problems. This approach has mixed results. Sometimes, users step up and contribute code, turning from consumers into collaborators. Other times, the issues remain open, waiting for when I can find time to address them. I try to help guide contributors through the process of making their first PR, but I can't always provide the level of support I'd like to.

The challenge of balancing my enthusiasm for these projects with the reality of limited time is something I'm still learning to navigate. I imagine even the most dedicated Open Source contributors face this tension, regardless of the size of their project.

Supporting Open Source Contributors

Through my experience on both sides of Open Source – as a user and as a contributor – I've gained perspective on how we can all better support the people who make this ecosystem possible.

If you want to support the Open Source ecosystem:

- Help out with PRs and documentation instead of just opening issues. Even small contributions can significantly lighten the load.
- Be patient when waiting for responses. Remember that most people are working on these projects in their limited spare time.
- Remember that there's a human behind every repository. Respectful communication goes a long way.
- Consider supporting financially if the project adds value to your work. Even small sponsorships can make a difference.

The sustainability of Open Source depends on recognizing the human aspect of software development. Behind every package, library, or framework is a person (or group of people) who have chosen to share their work with the world.

The Impact of AI

AI is already transforming Open Source development in ways I couldn't have imagined when I started programming. I use GitHub Copilot Review for quick PR summaries and Copilot itself for generating repetitive code and documentation. The "resolve issue with copilot" feature on GitHub is helping address long-open issues more efficiently.

These tools don't replace the need for human judgment and creativity, but they do reduce the friction of contribution. I'm grateful that GitHub offers Copilot for free for Open Source projects, as it's helping level the playing field between hobby projects and commercially-backed ones.

As AI continues to evolve, I see it amplifying the impact of individual contributors rather than replacing them. The human aspects of Open Source – community building, mentorship, vision setting – remain uniquely human endeavors.

The Future

As I look ahead, I'm optimistic about the future of Open Source. The model has proven its resilience and value over decades, and it continues to evolve. Open Source is about sharing and collaborating. It's about building something together, not just for yourself. It's about giving back to the community and helping others.

I wouldn't be where I am today without Open Source - I've learned from the community, made lifelong friends, and discovered what I want to do with my life. The skills I've developed through community organizing and project contributions have transferred directly to my professional work, making me a better engineer and collaborator.

As I continue this journey, I hope to keep growing communities, connecting developers, and contributing in whatever ways I can. The beauty of Open Source is that even small contributions add up to something meaningful over time. Whether it's organizing the next BoulderJS meetup, merging a PR to one of my projects, or helping a new developer make their first contribution, every action contributes to this global collaborative effort.

Let's make an awesome Open Source future together.

/ Patrick Heneise, April 28, 2025

@pradumnasaraf – Pradumna Saraf

github.com/pradumnasaraf
maintaine.rs/pradumnasaraf

Hey, I am Pradumna Saraf, an Open Source Developer/DevRel based out in India. I am also a Docker Captain, a DevOps and Golang Developer. I am passionate about Open Source and have mentored hundreds of people to break into the ecosystem. People in the community know me for the content I create on X (formerly Twitter) and LinkedIn, educating others about Open Source and DevOps tools. I also enjoy engaging with people in person and delivering talks at conferences.

My journey with Open Source began in 2021 during Hacktoberfest with the EddieHub Community (founded by Eddie Jaoude, shout out to him, I see him as my godfather and a good friend). When I joined the community, I was amazed by how people were helping each other and making their work public for anyone to see. At first, it felt a bit weird, but over time, I came to understand its importance. That spirit of openness and collaboration stuck with me, and here I am today, sharing my Open Source journey with you all.

Open Source: More Than Code

For me, Open Source is more than just code, it's a place where people from all over the world come together to work on projects, solve problems, and collaborate without being judged on their knowledge, race, colour, or anything else. You can learn and share your learning at the same time while making things better. And with it comes trust, transparency, and security.

Open Source Projects that I am involved in

I have been involved in many projects, especially in the DevOps space. I'm actively involved in CNCF projects, Docker, and various community projects, like the EddieHub Community. In these, I contribute to everything from documentation to DevOps tools implementation, depending on the project's needs. My key areas of contribution are Docker, Kubernetes, and CI/CD (mostly GitHub Actions). In addition, I also maintain a lot of my Open Source projects daily. You can check them on my GitHub.

Growing an Open Source community

To grow any community, the first and most important thing is creating a safe space where people feel comfortable talking and asking questions without hesitation. Having clear documentation, a README, and contributing guidelines helps people understand the project better. Also, having beginner-friendly issues helps people get started with the project much more easily. It's all about making them feel supported. Another key aspect is making contributors feel valued and giving them recognition, whether it's through regular shout-outs, mentioning them in different places like the README for their contribution, or just recognising their efforts. This encourages more people to get involved. Also, being vocal about the community on social platforms helps in building a strong, welcoming environment.

Maintainer's Challenges

There can be a variety of challenges that come with being an Open Source maintainer. One of them is keeping up with the volume of notifications — by notifications, I mean opening a new issue on the project, comments from contributors on pull requests, or just general questions and feedback. Sometimes it feels overwhelming and can lead to burnout. Another challenge is keeping the docs, README, guides, etc., updated as the project evolves — and in Open Source, things change fast. Other challenges can include funding, saying no to feature requests from the community, and sticking to the original scope of the project.

Supporting Maintainers

There are many ways one can support a maintainer. One of the most important things is following the contributing guidelines before jumping into contributing, because that contains everything from expectations, code styles, to prerequisites, and that saves a lot of maintainers' time if these things are followed. Another thing can be reviewing and triaging pull requests and issues, because the good thing about Open Source as a contributor is that you can do everything the same as a maintainer, you just can't merge the Pull Request. Last one is having some patience when you are contributing because a lot of maintainers are doing it voluntarily, and they have other full-time jobs and families to look out for, so there might be some delay in getting back to you.

Open Source Security: Practices and Challenges

Security is one of the areas where we need to do much more. Using the latest versions of dependencies that are vulnerability-free is a good start. GitHub has a bot called Dependabot that

automatically scans them, notifies you, and even creates a PR for you. Another important thing is that when you're using tokens like GitHub tokens, make sure you give them the least privileges needed. A Security Policy (SECURITY.md) is really important, as it allows contributors to report security vulnerabilities privately so they're not exposed to the public, which could make the project more vulnerable.

Not using the latest and greatest vulnerability-free dependencies can make the end product vulnerable to users — this especially happens with unmaintained projects that still have a large user base. Another issue is not having security audits or scans on pull requests. Specifically for containerised projects, using older versions of base images that contain CVES can make the whole system less secure. Last but not least, there's often a lack of security and compliance knowledge among Open Source contributors and maintainers. On the other hand, if your application uses containerization technologies, it's better to use image scanning tools like Docker Scout. You can automate that process with GitHub Actions to make sure your image is clean and CVE-free.

AI and Open Source

AI is both good and bad. On the one hand, it has improved development speed. Now developers can write and debug code much faster. It has also improved code quality and optimisation. On the other hand, as more code comes in, it becomes harder to review everything, and that can lead to major security issues, especially when people rely on AI tools for reviews. Also, with AI, it's getting harder to identify how much effort a contributor has put in; someone might take 15 days to implement a feature by writing everything by hand, while someone else could do it in 15 minutes using an AI tool. So, there should be a right balance to address this.

My two cents for current and new maintainers

First of all, if you are reading this and you are a maintainer, congrats and thank you for making this community great. A couple of tips that help me in my maintainer journey are having great docs, a clear README, and contributing guidelines — these will help you save a lot of upfront time and effort with new contributors. Automate as much as possible. Run linters and tests using tools like GitHub Actions so you don't have to manually do the repetitive task and can focus more on code and collaboration. Keeping security in mind, create a SECURITY.md file so that people can report vulnerabilities in private instead of exposing them publicly, which could make it more risky. One of the most important things is to set boundaries that carry your vision and goal of the project, because there are times when people will want to get new features added, but that's not how you have envisioned it. Saying "No" is the most reliable option. And there are some others, like labelling the issues well and getting more maintainers on board.

On the other hand, have some empathy for the new contributors because they might have less knowledge compared to a maintainer, so try to help them get started, encourage and uplift them.

At this point, I owe everything to Open Source. From whom I connected with to the opportunities I received for career growth, I will keep the momentum going and continue to support Open Source. At last, I am here writing this piece of content you are reading because of Open Source only.

You can connect with me on these channels:

- Personal Website: https://pradumnasaraf.dev
- Twitter (X): https://x.com/pradumna_saraf
- LinkedIn: https://www.linkedin.com/in/pradumnasaraf
- GitHub: https://github.com/Pradumnasaraf

@raisinten – Darshan Sen

github.com/raisinten
maintaine.rs/raisinten

Hi, I'm Darshan Sen, an award-winning Governance Member of high-profile Open Source projects such as Node.js and Electron. If you use these projects, there is a good chance you're relying on code I've authored. Currently, I work as a Technical Lead at Postman, where I continue to blend Open Source with professional work.

It all started with Open Source

When I was first introduced to Open Source in university, I found the concept very fascinating. The idea that anyone, anywhere, could improve a software that thousands or even millions use every day, really captivated me. At the same time, I was also drawn to C++, a language many of my peers feared due to its complexity. That challenge pushed me toward contributing something meaningful to a C++ heavy Open Source project, so I began my search.

Hello, Node.js

In the fall of 2020, I cloned Node.js and attempted to compile it on my family laptop which was running a 32-bit Ubuntu OS. It failed to compile, so I submitted my first C++ patch to add back 32-bit Linux support and the maintainers landed it! That validation motivated me to work on more things that caught my interest in Node.js and its dependencies like V8, libuv and OpenSSL. I was nominated and subsequently appointed to serve as a collaborator and a voting member of the Technical Steering Community (TSC) by the project community. I eventually became Node.js's 36th highest contributor out of 3,595 worldwide. To help grow the community, I provided one-on-one mentorship to numerous contributors and nominated and onboarded new collaborators. I'm best known for creating the Single Executable Applications feature in Node.js and leading the development of this strategic initiative which is critical to the success of the project. In recognition of my contributions, I received the "Outstanding Contribution from a New Arrival" award as part of the JavaScriptLandia Awards at 2022's OpenJS World. Beyond honing my technical skills, I found a supportive and inspiring community.

Hey, Electron

I later became involved in the Electron project, where I rose to become the 54th highest contributor out of 1,303 worldwide. Eventually I was nominated and accepted into Electron's Governance. My work there is primarily focused on performance. For example, I contributed a 60% speedup for a complex and long-standing Intel macOS issue that was slowing down the startup times in Google Chrome and all Electron-based apps.

Thank you, Open Source

Through Open Source, I've met some of my closest friends, which has led to exciting opportunities, including at Postman, the popular API Platform. As a Technical Lead, I work on Node.js and Electron related initiatives, mentor colleagues and help teams turn their goals into reality.

Open Source has not only shaped my career but also given me a sense of purpose, lasting friendships and a platform to give back to the developer ecosystem.

What's in it for you?

If you need my expertise, get in touch on my personal email for consulting. If you appreciate my Open Source work, here's your chance to support it directly.

Follow me on Twitter / X, LinkedIn and Bluesky for insights into Open Source and a glimpse of life through my lens. Follow me on GitHub for my Open Source projects. Follow me on YouTube to catch my talks. Let's connect!

@raphael – Raphaël Simon

github.com/raphael
maintaine.rs/raphael

Hi, I'm Raphaël Simon, the creator of Goa, which you can find at goa.design. It's a design-first framework that helps with building APIs and microservices in Go.

Goa began as a personal project with the simple goal of streamlining my workday and ensuring that our APIs were built correctly from the outset. It wasn't initially intended to be a community project.

Turns out, sometimes solving your own problems ends up helping a lot of other people too.

From one Rails app to fifty services

At the time, I was working at RightScale, which is now part of Flexera. The platform was growing from a single Rails application into a distributed system with over 50 microservices, running on hundreds of virtual machines.

This scaling led to a pretty common issue: inconsistent APIs.

Each service had its own style and little quirks. Nothing major on its own, but all together, it made the whole system tough to understand, integrate, and change. Once an API was live, it was almost impossible to fix without causing problems for customers or other parts of the system.

We really needed a better approach: designing APIs first, making sure they were consistent, and letting teams move quickly without creating headaches later on.

In Ruby, we had built Praxis to help with that. But as we were moving towards Go, I wondered if we could do even better.

After some trial and error, two main ideas came to light:

- **Completely separate the design from the actual implementation.**
- **Describe the API using a simple, expressive DSL written in Go itself.**

That's how Goa started. At first, it was just a solution for our team. I put it on GitHub mostly out of habit, not expecting much else.

The tweet that changed everything

For a while, Goa just sat quietly on GitHub with a few stars.

Then one day, Brian Ketelsen, who co-founded Gopher Academy and organizes GopherCon, stumbled upon it. He tweeted, “I think I want to marry the guy who wrote Goa.”

I wasn’t even on Twitter at the time. A coworker had to show me the tweet. It made me laugh and made me realize that maybe Goa was resonating with people beyond just our immediate needs.

After that tweet, plus some blog posts and mentions on Go Time, Goa’s GitHub stars jumped from about 10 to over 1,000 in just a few days.

That momentum brought in new contributors, community interest, and eventually an invitation to present Goa at GopherCon 2016.

Open Source has a way of surprising you, often when you least expect it.

Sharing work openly

Publishing Goa was always about sharing something I thought was genuinely useful.

My early experience with Linux showed me how powerful it is to work with systems you can understand, modify, and improve yourself. I carried those lessons with me, and when Goa came together, it felt natural to share it. Not because it was perfect, but because it might help others.

Open sourcing Goa also made it better. Real users brought new ideas, edge cases, and improvements that our internal team never would have found on its own.

In Open Source, the project grows beyond its creator. And that’s exactly how it should be.

What made Goa different

From the start, a few things were really important:

First, the generated code had to look natural, like something a human would write. Go developers really value clean, idiomatic code. If the generated code looked too mechanical or awkward, people wouldn’t use it.

Second, the DSL had to feel intuitive. Designing an API should be about describing real things — data structures, endpoints, payloads — without getting bogged down in technical details.

Finally, complexity needed to be hidden unless you wanted to dig deeper. Goa should make the hard stuff invisible most of the time, but still be understandable under the hood.

These early choices made a big difference and are still key to what makes Goa useful today.

Unexpected connections

One of the best surprises has been Goa's strong adoption in Japan.

The Japanese Go community picked up Goa early on, and some of the most important contributions came from developers like Taichi Sasaki and @ikawaha. They didn't just use the framework — they helped shape it.

It's a reminder that once you put something out there, you can't predict where it will end up or how far it will go.

Where Goa is today

Goa has come a long way since those early days.

Today, it generates complete scaffolding for HTTP and gRPC services, automatically creates OpenAPI documentation, and builds strongly typed client libraries that make working with APIs simpler and safer.

But the most important thing is that Goa is still evolving.

New features are constantly being added, driven by real-world needs from the community. Right now, for instance, we're expanding support for Server-Sent Events (SSE), making it easier to build Model Context Protocol (MCP) servers using Goa.

Projects like this keep Goa growing naturally, one practical improvement at a time, without losing sight of the original goal: to help developers design and build reliable APIs more easily and with less hassle.

Goa isn't just a framework you install once and forget. It's something you can grow with and help grow as your services change.

Lessons from the journey

Building and maintaining an Open Source project teaches you a lot more than you'd expect. Goa has taught me many things, including:

- Solve a real problem that you really understand.

- Make your project easy for people to get started with.
- Stay responsive, but also make sure it's sustainable for you.
- Value every contributor, no matter how big or small their contribution is.

And maybe most importantly, **be ready for the unexpected.** Sometimes a random tweet can open doors you didn't even know existed.

Building Goa has been one of the most rewarding experiences of my career. I'm thankful for everyone who found it, contributed to it, challenged it, and made it better.

I'm excited for what's next and for the new ideas that Goa will continue to inspire.

@sabderemane – Sarah Abderemane

github.com/sabderemane
maintaine.rs/sabderemane

I'm Sarah Abderemane, a software engineer at Kraken based in France.

I've been a contributor to some projects related to the Python language and Django framework and I'm a maintainer of the Django website for a few years.

You can find out more about me on my website and follow me on social media.

How I started to contribute

My journey began a few years ago with Hacktoberfest, the most popular project to start contributing to Open Source.

I contributed to a small project to explain part of a language like you're 5, for python language. After some time, I ended up reviewing others' issues. There was some issue in my submission for Hacktoberfest so I chatted with the maintainer to figure out the issue. At the end, I was offered to become a maintainer for the next year to help review the submissions and create new issues.

Then, I contributed to bigger issues and bigger projects with the creation of the read the docs of Jupyter accessibility repository and I finally contributed to Django website to implement dark mode on the website and I became a maintainer after this.

Vision and challenges

Open Source to me is a good way to share what we could have in common, improve something and give back to the community. It also means facing interesting questions and thinking about how to solve issues at our level.

As a maintainer we faced many challenges depending on the project: need of money to make some improvements, make sure to keep our existing users and attract new users. Also, keep in mind the diversity, to me it's important to have that to make people feel comfortable contributing because this is meant to be usable for everyone without exception.

All of that means that you have to keep a good balance with your personal life, which is not always simple. Find time for yourself, do sports, see friends, do extra activities... and have

enough time to review issues and PRs when you don't have time during the day. One thing you have to know is what your capacity is and set clear boundaries to avoid doing more than you can handle.

Maintaining a project can be challenging but it is a rewarding experience. You learn a lot. How to interact with people from other countries, how to structure the project so that it is understandable to everyone, etc. It's great to see so many people using what you have done or view folks having some interest in your project.

Community from the maintainer perspective

Building a community when you are a maintainer is not simple on a project which is big like Django. You have to follow the predefined rules and conform to the standards.

It's not easy to attract contributors and maintainers on a big project that have specific rules to follow, in order to keep all Django projects healthy and sustainable, but having a welcoming community is important.

One way we grow the community is via a mentorship program, Djangonaut Space, that helps folks to contribute to Django core and its ecosystem with a mentor and a supporter to help through the Open Source journey with things like impostor syndrome.

I hope that we have more people who contribute to Open Source and Django projects, and they will be the next generation to contribute and maintain those projects.

@saikrishna321 – Sai Krishna

github.com/saikrishna321
maintaine.rs/saikrishna321

Hi! I'm Sai Krishna (Director of Engineering, LambdaTest), an open-source contributor and maintainer with over a decade of experience. My journey began as an OSS user, overcoming the difficulties associated with mobile test automation. Witnessing firsthand how open-source tools empower the community sparked a deep interest in giving back. That's where my journey truly started. Since then, I've contributed to and maintained key projects like Appium, co-created tools like AppiumTestDistribution (ATD), and actively mentored the next generation of testers. For me, Open Source isn't just about code. It's about creating impact through community collaboration.

1. How did you get involved with Open Source?

Like many others, I started as an open-source user. As I relied on tools like Appium in my day-to-day testing efforts, I realized how these community-built tools made life easier for so many. That realization inspired me to contribute back. I began small, triaging issues and writing documentation, and gradually moved toward contributing code. Over time, I got involved in projects like Appium, Selenium, and ATD, eventually taking on maintainer roles to help shape the direction of the tools I once just used.

2. What's Open Source to you?

Open Source is a shared responsibility and a powerful force for innovation. It's not just about software. It's about enabling communities to solve problems together. Whether it's writing code, reviewing pull requests, improving documentation, or guiding a first-time contributor, Open Source thrives when we build with the community, not just for it.

3. What projects are you involved in?

- **Appium Java Client**: Focused on plugin architecture and improving parallel test support.
- **AppiumTestDistribution (ATD)**: Co-created to address parallel execution challenges in mobile and IoT testing.

- **Selenium / Taiko**: Contributed to enhancing developer ergonomics and cross-browser testing stability.
- **Appium Plugins**: Developed several plugins to simplify and extend Appium capabilities:
  - Appium Device Farm
  - Appium Gestures Plugin
  - Appium Wait Plugin
- **Appium Conference**: Active participant and advocate for sharing best practices and building global tester communities.

4. How do you grow your community?

- **Listen and Adapt**: ATD was shaped by community feedback, such as adding remote device execution support for distributed teams.
- **Simplify Onboarding**: Built tools like `appium-installer` to make setup easy for newcomers.
- **Accelerate Go to Market**: Appium Device Farm enables organizations to quickly set up scalable mobile device grids for automated testing. This helps improve device coverage, accelerate feedback, and reduce time to market.
- **Events and Outreach**: Conduct webinars, workshops, and community calls to connect with testers and developers.
- **Documentation First**: Clear and actionable guides are a top priority to help contributors ramp up quickly.

5. Main Challenges as a Maintainer

- **Burnout and Sustainability**: Balancing a full-time job with OSS commitments requires careful management.
- **Technical Debt**: Evolving architectures while keeping backward compatibility is complex.
- **Security Oversight**: Staying ahead of vulnerabilities in dependencies is a continuous effort.
- **Succession Planning**: Building a pipeline of contributors who can take over when needed is essential for long-term project health.

6. How Contributors Can Support Maintainers

- **Non-Code Help**: Contribute to documentation, triage issues, or mentor first-time contributors.
- **Sponsorships**: Encourage organizations to support OSS via GitHub Sponsors or Open Collective.
- **Be Proactive**: Submit detailed bug reports and well-crafted pull requests, and respect maintainers' time.
- **Security Mindset**: Assist with implementing reproducible builds, static analysis tools, and secure development pipelines.

7. Key Security Practices

- **Automated Scanning**: Integrated static analysis and dependency scanning into CI/CD workflows.
- **Manual Reviews**: Maintained strong code review practices to catch subtle vulnerabilities.
- **Reproducible Builds**: Adopted to ensure traceability and auditability across key projects.
- **Community Awareness**: Hosted regular sessions on secure coding and testing for mobile and web ecosystems.

8. Impact of AI on Open Source

- **Faster Innovation**: AI tools like code assistants and CI copilots boost productivity and accessibility in Open Source.
- **New Risks**: AI-generated code introduces concerns around licensing, bias, and reproducibility.
- **Upskilling**: Developers with AI and Open Source experience are in high demand. AI knowledge is becoming essential for modern maintainers.

9. Advice for Maintainers

- **Mentor Early**: Invest time in contributors who can grow into future maintainers.
- **Use Metrics**: Track pull request volume, response time, and contributor engagement to identify gaps.

- **Automate Everything**: Leverage bots, scripts, and CI tools to reduce fatigue and focus on strategic efforts.
- **Be Community-Driven**: Focus on solving real pain points. ATD succeeded by addressing everyday mobile testing challenges.

Final Thoughts

Open Source is built on invisible work: code reviews, mentorship, documentation, and triage. That work deserves recognition and support. As the landscape evolves and AI plays a bigger role, maintainers must innovate while staying sustainable and inclusive. The future of Open Source depends on shared responsibility and visible maintainership.

Connect with Sai Krishna:
GitHub: saikrishna321
Blog: saikrishna.tech LinkedIn: Sai Krishna

@samdark – Alexander Makarov

github.com/samdark
maintaine.rs/samdark

I'm Alexander Makarov maintainer of the Yii framework, and today I'm writing about my way in Open Source.

I graduated in 2007 from the Computer Science faculty and at that time I wasn't paying any attention to Open Source. I knew there were free software you're not paying for that some people did for whatever reason and there are paid software companies that did it for profit. I made some free software myself such as a cyber-club management system written in Delphi. I've posted it to some forums as a zip archive containing exe files but never thought that someone would ever want to look at the source code.

Around 2007 I was J2EE developer and I was searching for a cheap hosting to create my own blog. Java hosting was not cheap so I did a blog in PHP. First, without any framework. Then I've tried CodeIgniter and I liked it. It was simple and docs were awesome. I've co-created a community website about it, participated in docs translation, fixed some minor bugs. Then I've started wondering how does it work and found that I do not like many parts of the framework. I've tried to communicate with the framework team but then EllisLab, the organization behind it, was focused on other products so I was pretty much ignored. Disappointed, I've started looking elsewhere: Zend Framework, CakePHP... these were not as good as I wanted these to be... and then I've stumbled upon Yii. The website was full of low quality JPEG-s. It looked like something from 2001 but I've read the docs, tried it and it made perfect sense to me. In 2008 I've created another community, did translation and started to bring good things to Yii that I've seen in Java Struts, Spring, and, of course, CodeIgniter. By 2010 I was sending too many patch files (yeah, SVN and Google Code were still there) to review by core team so, instead, I was invited to join. Core team was awesome! After a year I've changed my opinion on Open Source overall and started to become a maintainer.

Nowadays Open Source to me is one of the ways I contribute to the world. I'm not really good at tolerating imperfection so sometimes I polish something or add a feature I need to products I use daily. And, of course, I'm maintaining the Yii framework. Doing that since 2010 and love it. I use it myself, I speak about it, I was invited to many places in the world to talk about it, knew great people because of it and love interacting with our team and community overall.

The community is driven by example. We are very different in the team but we keep doing high

quality stuff, are communicating well and overall creating a very healthy place to be in. The most active community members are becoming maintainers eventually.

There are, of course, difficulties:

1. Lack of time. We're involved in commercial projects so the time we can allocate to Open Source is quite limited.
2. Lack of funding. We'd like to buy yourself time to work on Open Source full time but that is only partially possible thanks to the foundation we've created. It's quite small so personally, I'm not using any of the funds. Team members do and I am glad that they've got more time this way.
3. Can't really plan. Because it's hard to predict what time is available for each team member and even for myself, any deadlines start to make almost no sense.
4. Sometimes critics are hard on maintainers.

To support your favorite product or maintainer you can:

1. Say "thanks!". That matters cause we're not getting it too often.
2. Provide constructive feedback.
3. Create issues.
4. Contribute with code.
5. Ask if another maintainer is needed.
6. Post about the product in question.
7. Contribute to a fund if there's any.

As the project grows, it becomes used by thousands of projects. Security becomes very important. At Yii we can't afford a bounty program but we've defined ways to communicate with security researchers. We use GitHub's advisories feature flow to report and fix issues discovered. Also, we deep-dived into security ourselves to make less mistakes. A good place to start learning is OWASP.

Recently AI/LLM started to be on hype. We're checking current trends and I'm really glad that we've decided to pursue 100% code coverage, 100% mutation score and near 100% type coverage. That allows us to leverage coding agents to improve our code. For example, we did some experiments about enhancing performance of core libraries and accepted half of the changes. All that thanks to tests. Coding agent worked a few days without stopping trying things, hallucinating, breaking code, fixing code, starting from scratch and repeating again and again.

As for projects without tests, I'm in the skeptics camp. At least for now. LLMs are being improved constantly and coding agents are as well. MCP gives more and more context. So in

a year or two we might see a breakthrough in quality: less hallucinations, more quality, more "understanding". That would still require at least a single visionary engineer to lead the project.

If you're a maintainer, don't hesitate to ask for help, communicate with people and enjoy the process. People in Open Source are amazing.

@sanjayaksaxena – Sanjaya Kumar Saxena

github.com/sanjayaksaxena
maintaine.rs/sanjayaksaxena

Open Source is not just about code availability—it is a powerful philosophy of collaborative innovation. To me, as a builder, tinkerer, and maintainer of the winkJS project in Javascript and its flagship library winkNLP, Open Source represents freedom with responsibility. My builder instinct thrives in the freedom to create transparent solutions, my tinkerer spirit relishes the ability to continuously improve what others have started, and my maintainer mindset embraces the responsibility to ensure security, performance, and reliability for all users.

Beyond popularity, industry recognition or compliance achievements, what drives me is the opportunity to advance collective knowledge rather than serve corporate interests alone—creating tools that honor diverse skills and contributions equally in the process.

Proprietary Roots to Open Source

My engagement with Open Source began when I was in my 50s, and it contrasts with my early years in software development. Starting my career in the early 1980s in India, I found myself in a setting that was dominated by closed-source, proprietary systems.

The turning point arrived in the 2010s during my involvement in analytics and NLP projects, notably one designed to assist Indian farmers. Presenting our analytics-driven NLP research at international conferences made me question why we were limiting these innovations to academic circles instead of Open Sourcing them—a realization that fundamentally transformed my approach to technology development.

Inspired by this shift in perspective, the core team members at the organisation I co-founded—Prateek, Rachna and I—began to wonder, "Why not Open Source?" This simple yet powerful question sparked a journey toward creating the Open Source project winkJS.

License, Philosophy, and the True Meaning of "Open"

Initially, we Open Sourced smaller NLP and machine learning utilities under the AGPLv3 license, before ambitiously moving forward to develop an integrated NLP tool—winkNLP. Soon, however,

we were confronted with a philosophical dilemma: the somewhat restrictive copyleft nature of AGPLv3 inadvertently limited the freedom of developers, contradicting our initial purpose.

An intense internal debate about what "open" genuinely means led us to a critical realization: genuine contribution to Open Source has to originate from within—it can't be mandated through restrictive licensing. In a pivotal moment of clarity, winkJS transitioned to the MIT license, embracing a philosophy that genuinely aligned with our vision.

This wasn't merely a licensing change—it was a commitment to trusting the community, honoring collaboration, and embodying the true spirit of Open Source.

Code with Conscience

Our MIT license adoption coincided with an unwavering commitment to development practices prioritizing reliability and security. From day one, we established non-negotiable standards: 100% test coverage, comprehensive static analysis, and thorough documentation.

This rigor proved invaluable when the tests caught a critical security issue known as Regular Expression Denial of Service (ReDoS)—a vulnerability where malicious inputs can drastically slow down or halt software—in one of the regular expressions used by winkNLP's tokenization engine, a core component responsible for breaking text into meaningful units.

Our coding guidelines addressed everything from basic security practices—prohibiting eval() and mandating Object.create()—to sophisticated protections against ReDoS attacks. Perhaps our most consequential decision was eliminating external dependencies entirely from winkNLP, dramatically enhancing security while enabling precise performance optimization.

Contributors embraced these standards with impressive dedication. One even went so far as to refactor their implementation multiple times and add thorough tests—demonstrating a level of care and commitment that reflects the values we strive to uphold.

After releasing winkNLP, discovering the OpenSSF Best Practices Guidelines was a key moment. They helped us refine our processes, and their principles now guide all our Open Source work.

And the Journey Continues

The work continues today with winkComposer—a real-time streaming-analytics framework. Just as our earlier work in NLP aimed to democratize language processing, winkComposer seeks to transform how developers work with continuous streams of data. The philosophy remains consistent: create tools that are open, reliable, and useful.

Processing millions of tokens per second without dependencies taught me lean design; that experience now shapes my streaming engine. This evolution reflects my own growth as a

developer. The lessons learned from building secure, dependency-free libraries guide how I approach streaming analytics—creating lightweight, modular components—robust enough for finance yet lean enough for IoT gateways.

What excites me most is how winkComposer combines statistical methods, narrow AI with knowledge graphs and Open Source LLM powered reasoning system, bringing the analytical power once reserved for batch processing into the streaming world using Node.js. We're building it with a focus on high-performance and reliability while embracing Responsible AI principles and OpenSSF security standards—another step in our mission to make technology accessible through Open Source.

Final Thoughts

Embracing Open Source has instilled in me an appreciation for what it means to have freedom with responsibility, and it has been inspirational to witness how community engagement amplifies innovation. My advice to maintainers, both experienced and new, is straightforward: embrace openness, rigorously uphold your project's standards, and trust the community. High standards are never barriers; rather, they inspire trust and collective excellence.

Ultimately, Open Source isn't just about writing code—it's about shaping an equitable technological future, collaboratively and transparently. I warmly invite anyone who shares this vision to join, contribute, and build alongside us—join winkComposer's discussions or write to me directly. Together, our collective efforts can create solutions that transcend individual capabilities and genuinely serve the community.

Contact information

- https://github.com/sanjayaksaxena
- http://winkjs.org/

@shazow – Andrey Petrov

github.com/shazow
maintaine.rs/shazow

My first big Open Source project was urllib3. Today it's used by almost every Python user and receives about a billion downloads each month, but it started in 2007 out of necessity.

I was working at TinEye (formerly known as Idée Inc.) as my first "real" job out of university, and we needed to upload billions of images to Amazon S3. I wrote a script to get processing and estimated how long it would take to finish... two months! Turns out in 2007, HTTP libraries weren't reusing sockets or connection pooling, weren't thread-safe, didn't fully support multipart encoding, didn't know about resuming or retries or redirecting, and much more that we take for granted today.

It took me about a week to write the first version of what ultimately became urllib3, along with workerpool for managing concurrent jobs in Python, and roughly one more week to do the entire S3 upload using these new tools. A month and a half ahead of schedule, and we became one of Amazon's biggest S3 customers at the time.

I was pleased with my work. I was just months into my role at TinEye and I already had a material impact. Reflecting on this time almost two decades later, I realized that doing a good job at work was not what created the real impact. There are many people out there who are smarter and work harder who move the needle further at their jobs than I ever did.

The real impact of my work was realized when I asked my boss and co-founder of TinEye, Paul Bloore, if I could Open Source urllib3 under my own name with a permissive MIT license, and Paul said yes. I did not realize at the time how generous and rare this was, but I learned later after having worked with many companies who fought tooth and nail to retain and control every morsel of intellectual property they could get their hands on.

It's one thing to write high impact code that helps ourselves or our employer, but it's another thing to unlock it so that it can help millions of other people and organizations too.

Choosing a permissive Open Source license like MIT made Paul's decision easy: There was no liability or threat to the company. TinEye had all the same rights to the code as I did or any other contributor did. In fact, Paul allowed me to continue improving it while I worked there because it benefited TinEye as much as it benefited everyone else.

Releasing urllib3 under my own name allowed me to continue maintaining and improving the project even after leaving, because it was not locked under my employer's namespace and I felt more ownership over the project.

Hundreds of contributors started streaming in, too. Nobody loves maintaining a fork if they don't have to, so it's rational to report bugs upstream and supply improvements if we have them.

The growth of urllib3 since the first release in 2008 has been a complicated journey. Today, my role is more of a meta-maintainer where I support our active maintainers (thank you Seth M. Larson, Quentin Pradet, Illia Volochii!) while allowing people to transition into alumni maintainers over time as life circumstances change. It's important to remember that while funding Open Source is very important and impactful (please consider supporting urllib3), it's not always about money. People don't want to work on one thing their whole life, so we have to allow for transition and succession.

I learned many lessons from my first big Open Source project, and I continue to apply them to all of my projects since then with great success. I hope you'll join along!

/ Andrey (shazow.net)

@skywinder – Petr Korolev

github.com/skywinder
maintaine.rs/skywinder

My journey into Open Source began over a decade ago when I created the GitHub Changelog Generator.

At that time, it was my first real project written in Ruby — and I couldn't have imagined how much it would resonate with the community. The project quickly gained momentum, reaching over **6,000 stars**, becoming one of GitHub's trending repositories back then, which felt truly incredible.

As my career evolved and my focus shifted to other projects and various new initiatives — the Changelog Generator was gracefully passed into new hands.

I'm especially grateful to Olle Jonsson, who picked up the project and took it to a new level: improving the codebase, making it more professional, polished, and sustainable.

Olle's dedication gave the project a second life, and it's one of the most beautiful feelings in Open Source — seeing a project grow beyond yourself, like raising a child who eventually becomes independent and thrives on their own.

Here's a glimpse of the journey captured in our contribution history: https://github.com/github-changelog-generator/github-changelog-generator/graphs/contributors

I'm deeply thankful to every contributor who helped shape this project, adding ideas, bug fixes, improvements, and trust.

Maintaining and contributing to Open Source is not just about writing code; it's about creating communities, fostering growth, and building something lasting together.

It's been one of the most fulfilling parts of my journey as a developer and human being.

How did you get involved with Open Source?

I've been involved in Open Source for over 10 years. It all started when I began creating tools for my own needs, and soon these projects found broader adoption. Seeing how my work could help others motivated me to contribute not just to my own projects, but to many other Open Source efforts as well.

What's Open Source to you?

For me, Open Source is about freedom, community, and shared progress. It's a space where collaboration fuels innovation, and where everyone has the power to improve and build on each other's work.

What projects are you involved in?

I've created several popular Open Source projects, including:

GitHub Changelog Generator — a tool to automate changelog creation

web3swift — a Swift library for interacting with Ethereum

ActionSheetPicker — a widely used iOS UI component

I also actively contribute to OMI, an innovative wearable AI hardware project that I'm very excited about right now.

How do you grow your community?

By being welcoming, responsive, and maintaining clear contribution guidelines. Good documentation, a friendly tone, and recognizing contributions all help to organically build and sustain a healthy community.

What are the main challenges you face as a maintainer?

One challenge is users who request features or fixes but aren't willing to contribute themselves — sometimes disappearing after raising issues. Another growing challenge is the influx of AI-generated low-quality pull requests. While automation can help, it also creates a lot of noise and extra review work, often without meaningful contributions.

What are some ways contributors can better support maintainers?

Take the time to read the contribution guidelines carefully

Submit thoughtful, high-quality pull requests

Say "thank you" — appreciation goes a long way!

Where possible, support the project through sponsorship or donations

What are some of the key security practices you've implemented in your project?

One key practice is maintaining clear policies for handling sensitive data — such as never committing environment variables or secrets to repositories. Simple but crucial. Additionally, having well-defined contribution and review processes helps catch potential issues early.

What do you think are the biggest security challenges facing Open Source today?

Commitev env vars security remains a major concern.

What's the impact of AI on Open Source development?

AI has a double-edged impact. On the positive side, it can accelerate contributions and help automate tedious tasks. But it also brings a surge of low-effort, machine-generated pull requests that maintainers must sift through, which adds noise and maintenance burden.

What advice would you give to current and new maintainers?

Stay open, be kind, and treat your contributors the way you would want to be treated. Clear communication, patience, and setting a strong but welcoming project culture are the foundations for long-term success.

@srinivasantarget – Srinivasan Sekar

github.com/srinivasantarget
maintaine.rs/srinivasantarget

Hi! I'm Srinivasan Sekar, Director of Engineering at LambdaTest and an Open Source contributor/maintainer. My journey began 8–9 years ago when I struggled with flaky mobile test automation pipelines in Appium. Determined to fix this, I contributed my first patch to improve its CI system—and discovered the power of collaborative problem-solving. Today, I maintain tools like Appium Java Client, co-created AppiumTestDistribution (ATD) for parallel testing, and lead the Appium Conference. Beyond code, I mentor testers, blog at srini.codes, and build plugins to simplify workflows.

For me, Open Source means "building with, not just for" the community.

---

1. How did you get involved with Open Source?

As an Appium user, I faced constant CI failures during mobile test runs. Instead of waiting for fixes, I dove into the codebase to stabilize the pipelines. That first success seeing my contribution benefit others hooked me. Over time, I expanded my work to projects like Selenium, Taiko, and ATD, transitioning from contributor to maintainer to member as I focused on systemic issues like parallel testing bottlenecks.

---

2. What's Open Source to you?

Open Source is a **force multiplier** for innovation. It's about democratizing technology, fostering collective ownership, and empowering communities to solve real-world problems. For me, it's not just code, it's mentorship, documentation, and creating tools like ATD to simplify complex workflows for thousands of developers globally.

---

3. What projects are you involved in?

- **Appium Java Client**: Enhanced parallel testing capabilities and plugin ecosystems.
- **AppiumTestDistribution (ATD)**: Co-founded to solve parallel execution challenges in mobile testing, later expanding to IoT.
- **Selenium/Taiko**: Contributions to test frameworks and community-driven tooling.
- **Appium Conference**: Co-Founded to unite global testing enthusiasts and share best practices.
- **Appium Plugins:** Developed a bunch a Appium plugins to help ease development efforts
  - Appium Device Farm
  - Appium Gestures Plugin
  - Appium Wait Plugin

---

4. How do you grow your community?

- **Listening First**: ATD evolved based on user feedback (e.g., remote device execution for distributed teams).
- **Accessible Onboarding**: Created plugins like appium-installer to simplify environment setup.
- **Events & Advocacy**: Hosting conferences, AMA sessions, and workshops to engage testers globally.
- **Documentation**: Prioritized clear guides to lower entry barriers for new contributors.

---

5. Main Challenges as a Maintainer

- **Burnout & Sustainability**: Balancing maintenance with full-time roles, especially as projects scale.
- **Technical Debt**: Evolving architectures (e.g., Appium's shift to plugin-based systems) while maintaining backward compatibility.
- **Security Oversight**: Managing vulnerabilities in dependencies
- **Succession Planning**: Ensuring continuity when key contributors step down.

---

6. How Contributors Can Support Maintainers

- **Non-Code Contributions**: Improve documentation, triage issues, or mentor newcomers.
- **Financial Backing**: Advocate for corporate sponsorships or Open Collective funding.
- **Proactive Engagement**: Submit reproducible bug reports and respect maintainers' time.
- **Security Advocacy**: Help implement SAST/SCA tools and reproducible builds.

---

7. Key Security Practices

- **Automated Tooling**: Integrated SAST (Static Application Security Testing) and dependency scanning into CI/CD.
- **Manual Code Reviews**: Maintained a good manual review rate to catch nuanced vulnerabilities.
- **Reproducible Builds**: Adopted in almost all of the projects to ensure traceability.
- **Community Education**: Hosted sessions on secure testing practices for mobile ecosystems.

---

8. Impact of AI on Open Source

- **Accelerated Development**: Tools like Cline, Roo integrated with Models like Claude, etc increases productivity and reduce costs which enable SMEs to innovate.
- **New Challenges**: Increased security risks (e.g., IP infringement, model bias) requiring guardrails.
- **Skill Demand**: Most developers value Open Source AI experience, creating upskilling opportunities.

---

9. Advice for Maintainers

- **Build a Bench**: Actively mentor successors to reduce risks.
- **Leverage Metrics**: Track Contributor Confidence and PR distribution to identify gaps.
- **Automate Relentlessly**: Reduce fatigue through CI/CD pipelines and bot-assisted triage.
- **Stay Community-First**: Prioritize user needs over "shiny" features—ATD's success stemmed from solving real-world parallel testing pain points.

Final Thoughts

Open Source thrives on **visible maintenance** and **shared responsibility**. As AI reshapes our ecosystem, maintainers must balance innovation with sustainability—whether through smarter funding models or inclusive community design. Let's make the invisible work of maintainers *seen* and *supported*.

Connect with Srinivasan:

- GitHub: SrinivasanTarget
- Blog: srini.codes
- LinkedIn: Director of Engineering, LambdaTest

@sy-records – Lu Fei

github.com/sy-records
maintaine.rs/sy-records

I'm Luffy, a full-stack developer passionate about Open Source. You can find me on GitHub @sy-records.

Before getting involved with Open Source, I was just an ordinary software user and a PHP developer, knowing little about the communities, collaboration, or the idea of contributing behind the scenes.

As time passed, I gradually realized that Open Source is not just a collection of technologies — it's also an extension of values and spirit.

I created projects like Simps and PHPMQTT, and actively contributed to several open-source projects including Docsify, Hyperf, Swoole, Typecho, and Apache Answer.

From being a user to becoming a contributor, and eventually a maintainer of Open Source projects, this journey has profoundly changed my career path and shaped my understanding of the tech world.

How did you get involved with Open Source?

In 2019, I joined the Swoole open-source project and began working on a documentation overhaul.

This involved not only redesigning the front-end UI but also correcting outdated content and adding more sample code.

During the process, I encountered some issues: the new documentation was built using docsify, which at that time had some long standing search-related problems, such as:

- Unable to search content within tables
- Incorrect scroll position when clicking search results
- Inability to search list content

I started by submitting issues to the docsify repository to see if the maintainers could address them.

After receiving responses, I found time to submit PRs with fixes, which were fortunately merged smoothly.

Through continued contributions to docsify, I was eventually invited to join the core team, and today, I am proud to be an owner of the docsify project.

After completing the Swoole documentation overhaul, I also helped promote docsify — leading to its adoption by organizations like Hyperf, OpenMix, and ApolloConfig for their documentation needs.

As my understanding of Open Source deepened, I gradually moved from simply fixing bugs to contributing code and improving documentation across projects, eventually becoming a maintainer for several of them.

What's Open Source to you?

For me, Open Source is both a passion and a beginning.

It's not just a technical practice — it represents a set of values.

It stands for openness, sharing, and collaboration.

By participating in Open Source, I have not only improved my technical skills but also met many like-minded friends.

More importantly, Open Source has given me a profound sense of belonging and achievement.

What projects are you involved in?

Recently, I've been active in the Apache Answer project.

In addition, I continue to contribute to projects such as:

- laravel/octane: Supercharge the performance of your Laravel application.
- hyperf: High-performance coroutine framework based on PHP Swoole.
- docsify: A magical documentation site generator.
- typecho: A simple yet powerful blogging platform based on PHP.
- simps/mqtt: MQTT Protocol Analysis and Coroutine Client for PHP. Supports versions 3.1, 3.1.1, and 5.0 of the MQTT protocol, including WebSocket support.

Since PHP was the language that started my journey, I also contribute to the PHP ecosystem, including websites and Chinese documentation translations.

What are the main challenges you face as a maintainer?

As a maintainer, I face several challenges:

- Time Management: Balancing project maintenance with a busy work and personal life schedule.
- Community Management: Handling feedback from community members and ensuring the community remains active and healthy.

Contributors or maintainers may sometimes move on due to changes in their work or technical interests.

Taking docsify as an example — even though core maintainers are still around, the update frequency and responsiveness have slowed compared to a few years ago, making it difficult to release new versions.

I sincerely hope that more fresh contributors will join and help revitalize the docsify project.

Incidentally, I'm also pushing for a new release of docsify in Maintainer Month, which can be experienced at https://preview.docsifyjs.org/.

Currently, I make time during my workday to check GitHub notifications and emails from Gmail, trying to balance my main job while also staying active in the open-source community.

Final Thoughts

From an ordinary user to a project maintainer, this journey has brought me growth, friendships, and a sense of accomplishment.

I am deeply grateful to the open-source community for the support and opportunities it has given me.

I look forward to continuing this journey alongside more passionate and like-minded friends.

@thomaspoignant – Thomas Poignant

github.com/thomaspoignant
maintaine.rs/thomaspoignant

I am Thomas Poignant, living in Paris, France and I am writing about my Open Source journey.

I am currently Head Of Engineering in Leboncoin *(one of the largest classified ad marketplace in Europe),* and I've worked in the software engineering field for more than 16 years.

I started with Open Source like many of us by using it, but I always had in mind that I wanted to contribute back to the awesome ecosystem that Open Source provides. And when I moved to a more leadership position in my day job, putting me away from the code more and more, it was obvious for me that I will be involved more in Open Source.

I started with a small library called scim-patch, and now I spend most of my time around feature flags I am building GO Feature Flag, a feature flag platform that works with all your favorite languages and is integrated easily in any company tech. You can start using feature flags super fast with GO Feature Flag.

And I am also part of the OpenFeature technical committee. OpenFeature provides an open specification that provides a vendor-agnostic, community-driven API for feature flagging that works with any management tool or in-house solution.

What are the main challenges you face as a maintainer?

Honestly, the biggest pain for me is just **finding the time** outside of my regular job to actually do all the stuff I've got planned for the project. It feels like there's always a backlog in my head!

Another thing that's a bit of a downer is **not really knowing who's using the thing I pour my energy into**. As a maintainer, you rarely get a good sense of your project's impact. It's kind of weird not knowing who's finding it useful or what they like about it. You just keep plugging away hoping it's helping someone out there.

How do you grow your community?

You've got to put people first, right? If what you're building isn't a good experience, it's tough to get anyone to stick around, let alone contribute. That's why with GO Feature Flag, I really focus on making it the best tool it can be, with clear docs and helpful support. People need to dig what you're doing before they'll jump in to help.

After that, it's all about being welcoming. I try to be super polite and positive with everyone who uses the project, always trying to sort out any problems they run into. Making people feel welcome is huge.

But yeah, getting a *contributing* community going is a different beast. You see folks pop in to fix their own little itch, which is awesome! But getting people to commit to the bigger picture, to really dive deep and contribute in a substantial way? That's a tough nut to crack. People have their own stuff going on, and it takes a lot for someone to invest the time to really understand a complex project.

What are some of the key security practices you've implemented in your project?

Security is a big deal, and we've baked it into our workflow in a few key ways. First off, **Dependabot keeps our dependencies on their toes with weekly updates**, so we're not lagging behind on crucial fixes.

We run security checks on the Docker images and Helm charts we are building to catch any potential vulnerabilities early on.

For ongoing monitoring, **we've integrated Snyk** into our processes. It helps us continuously scan for and address security issues.

Finally, to balance security with community contributions, **we hold off on running CI pipelines for first-time contributors**. This gives us a chance to review their changes before automated processes kick in.

What do you think are the biggest security challenges facing Open Source today?

"One of the biggest security challenges facing Open Source today is the increasing prevalence of supply chain attacks, where malicious code is injected into widely used projects. The recent compromise of a popular GitHub Action, `tj-actions`, (CVE-2025-30066) is a stark reminder of this threat.

Attackers are targeting the open-source ecosystem because its interconnected nature allows a single successful attack to have a widespread impact. Open Source projects are as vulnerable as

any other software, and we need to be very vigilant about these kinds of attacks

What's the impact of AI on Open Source development?

AI is having a significant and multifaceted impact on open-source development. On the one hand, it's accelerating development workflows in exciting ways. This can be a huge benefit, especially for smaller open-source projects with limited resources.

However, this increased speed can come with a downside. There's a real risk of a decrease in code quality if AI-generated contributions aren't carefully reviewed. I've also observed instances where AI-generated code introduces new issues or requires significant rework, ultimately wasting the time of both contributors and maintainers.

What advice would you give to current and new maintainers?

My advice to both current and new maintainers is: Do it! Open Source is an incredibly exciting and rewarding journey. You'll benefit immensely from the feedback and help you receive from the community, and you'll find that the experience directly contributes to your growth and skills in your day-to-day work.

Conclusion

In closing, remember that Open Source is a journey. You can start small, gradually increasing your impact over time.

It's also important to keep in mind that success isn't always measured in GitHub stars. From my own experience, a project with a modest star count (28 stars) can still have a significant impact, reaching nearly 3 million downloads.

If you're looking to get involved and contribute in Open Source, please feel free to join me in building the GO Feature Flag at https://github.com/thomaspoignant/go-feature-flag. We're always happy to welcome new contributors!

- GO Feature Flag Website: https://gofeatureflag.org
- LinkedIn: https://www.linkedin.com/in/poignantthomas/
- GitHub: https://github.com/thomaspoignant
- Bluesky: https://thomaspoignant.bsky.social

@vdemeester – Vincent Demeester

github.com/vdemeester
maintaine.rs/vdemeester

I've always wanted to do Open Source, since I discovered it around 1999. The first Linux distribution I used was Red Hat Linux, and that was also my first encounter with Linux and Open Source. Little did I know that 20 years later I would work for the company that kind of introduced me to it!

I initially started looking into small projects I would need in my first startup job, like `fog`/`fog`, but what really got me started was the Docker project. Docker organized those "contribute meetups" in different places around their "birthday" and so, I made my first pull-request to the project that way. And from that point on, I was hooked. I also got involved a bit in the Archlinux User Repository when I was using it and now in NixOS.

What's Open Source to you?

For me, Open Source is "the way" to do things. I truly believe in sharing commons and the fact that we do better when we work together, in the open. And Open Source embodies that.

What projects are you involved in?

I am involved in a bunch of projects, some more than the others, but the highlights are:

- The TektonCD project, where I am one of the main contributors and a governance member. I am also the architect of our product based on top, OpenShift Pipelines.
- The Moby project and the Docker project. I have been a main contributor for years, and I am now a little less active.
- The Traefik project, where I was one of the first contributors—well, the second.
- The NixOS project, where I am maintaining a bunch of packages.

Nurturing Communities and Overcoming Hurdles

How do you grow your community?

This is a hard question, but essentially, to help a community grow, you need to make it welcoming to anyone, no matter what their background or interest is. And then you need to keep it alive. You need to make sure the goal(s) of the community are clear and drive the community to draw a path towards those goals. There are always some ups and downs during the life of a community—people leaving, new people coming—so you need to ensure that the community is standing for itself.

What are the main challenges you face as a maintainer?

I guess time management is one. As a maintainer, you need to triage user requests and proposals. You need to make sure you treat everyone equally. It can be a challenge to figure out what the priorities are.

Another challenge is around process. To be functioning, a community needs some process in place, but there is a balance to find. Too much process can become a burden and kill motivation and creativity. Not enough can make it very hard to make decisions. Also, maintaining those processes and making them evolve as the community evolves can be overlooked.

What are some ways contributors can better support maintainers?

Even seemingly minor contributions can be incredibly impactful. For instance, submitting pull requests that are unambiguous and thoroughly documented is a massive help. Likewise, ensuring changes are tested beforehand, contributing to documentation—whether by writing new material or updating existing content—and assisting with the initial review and categorization of issues are all activities of immense value.

Beyond individual code submissions, ongoing engagement from contributors is also key. When someone takes responsibility for a specific area, like a plugin or a particular feature, diligently acts on feedback, or offers support to other community members, it significantly eases the burden on maintainers. Ultimately, Open Source thrives on teamwork and a sense of collective ownership, which is vital for its long-term health.

Security in the Open Source Realm

What are some of the key security practices you've implemented in your project(s)?

In most projects I have been involved in, we are trying to apply all the security best practices we can find that make sense. That goes from linting and code scanning to automating dependency security fixes updates. But aside from tooling, these days, one area of focus for me is to look for ways to reduce dependencies in our project. Fewer dependencies usually mean fewer problems.

What do you think are the biggest security challenges facing Open Source today?

One of the primary difficulties arises from how extensively Open Source is now used. It's not uncommon for a project maintained by a very small team, sometimes just a few volunteers, to be a critical component in thousands of live production systems. This mismatch places enormous pressure on maintainers to deliver high-level security, often with insufficient resources.

Furthermore, the way we handle software dependencies presents another significant area of concern. A single security flaw in an upstream library can cascade through the ecosystem, impacting hundreds or even thousands of projects that depend on it. To effectively manage this interconnectedness and its risks, we really need to see advancements in tooling, the establishment of more rigorous processes, and a much stronger emphasis on collaboration between different ecosystems.

The Horizon: AI and Open Source

What's the impact of Artificial Intelligence on Open Source development?

I am not sure it is clear yet what impact Artificial Intelligence (AI) will have on Open Source development. I feel it can help tremendously maintainers, for example for triaging issues, reviewing code. But it also feels like a double-edged sword, and we already saw some AI-generated spam issues and pull-requests on some of our projects. With the "vibe" coding trend and agents, it feels there could be some enhancement in productivity, but I am a bit concerned about quality, and thus it's definitely something to be careful about.

@wasiqb – Wasiq Bhamla

github.com/wasiqb
maintaine.rs/wasiqb

Hello Open Source community! I'm Wasiq Bhamla, a Software Quality Assurance Engineer for more than 18 years and Open Source maintainer and contributor for almost a decade. I am passionate about building Testing tools and frameworks that empower the QA community.

After I complete my job time or on my days off from work, I dive straight in to work on my Open Source projects.

How did you get involved with Open Source?

It started in 2015 when I was given a task to automate Android and iOS applications. When I checked in my organization if there are any Test automation frameworks available in any other project teams, I found that there is none. So I created a Mobile Test automation framework, and thought to make it available for the world who might face similar challenges in using Appium directly by open sourcing that project.

Later on, whenever I face new challenges, I make sure to create a working simple and easy to use solution and make it Open Source on GitHub. In short, I started as the maintainer instead of a contributor.

However, there were times where I also contributed to other Open Source projects which I was using, but those contributions were very little. I mostly was involved with my personal frameworks and tools which I had created.

What's Open Source to you?

For me, Open Source is a way to help the community of Quality assurance Engineers around the world with the common problems which I face by creating a useful solution for everyone which is very easy to use and maintain.

What projects are you involved in?

Let me share details about few of my projects which I am maintaining:

Boyka Framework

Ultimate Test automation framework for automation of Web, API, Android and iOS applications. This project was created to simplify and ease the Test automation process for the QA community. It was implemented by using all my 15+ years of experience and learnings so the problems which I faced, this framework would help me and all the QA's overcome it.

Boyka_CLI

Boyka Framework command line assistant. This project was created for further simplifying the process of creating new or configuring the existing Boyka Framework projects.

Multiple_Cucumber_HTML_Reporter

Generate beautiful Cucumber HTML reports This project was created by another Open Source contributor named Wim Selles, who was looking for someone to adopt this awesome tool project. I had the privilege to take over the ownership of this project and become a new maintainer for the project.

Now it is nearing almost 1M downloads per month on NPM;

Maven_Publish_Action

Publish your Java JAR file to Maven Central with GitHub Actions. This project came into existence when the other Maven publish action which I was using, was not being maintained for almost a year.

I took the opportunity to completely rewrite the project in Typescript from the original Common JS. Now it is being actively maintained by me.

Main challenges as a maintainer?

The main challenge I have experienced as a maintainer, is lack of contributions. It becomes very hectic for me as a maintainer and core contributor to do each and every thing for my project, from adding new features to add documentations, updating readme, setting up the CI workflows, creating contents for the project to raise awareness for the projects. It sometimes causes burnout.

Another challenge is the lack of financial support for the Open Source projects. This however does not demotivate me, but it would really help in attracting new contributions which would highly boost the project's productivity.

How can contributors support maintainers?

Contributors can contribute to any project by helping fix any open issues, fix any documentation typos, triage any issues, and also help answering to queries raised by other users. There are plenty of ways where contributors can be of help to the maintainers.

Anyone can become a contributor, they just need to support the projects which they are using in their daily work. All you need to do is look where you can contribute and support the maintainer. You can even reach out to the maintainers who will be very happy to guide you and help you get started.

Key security practices in your projects?

In all my projects, I have set up dependabot to automatically update the vulnerable dependencies. I also make sure I have enabled all the GitHub Security scanning which would alert me whenever there is any security issue in any of my projects. These features from GitHub also include scanning of any accidental commit of any secure and confidential keys.

Advice to maintainers?

My advice to maintainers, whether new or experienced, is to always keep up the good work you are doing. It does not matter if there are times where you work for free, tirelessly and put in your extra precious hours, the thing that matters is what impact your project is having in your community.