

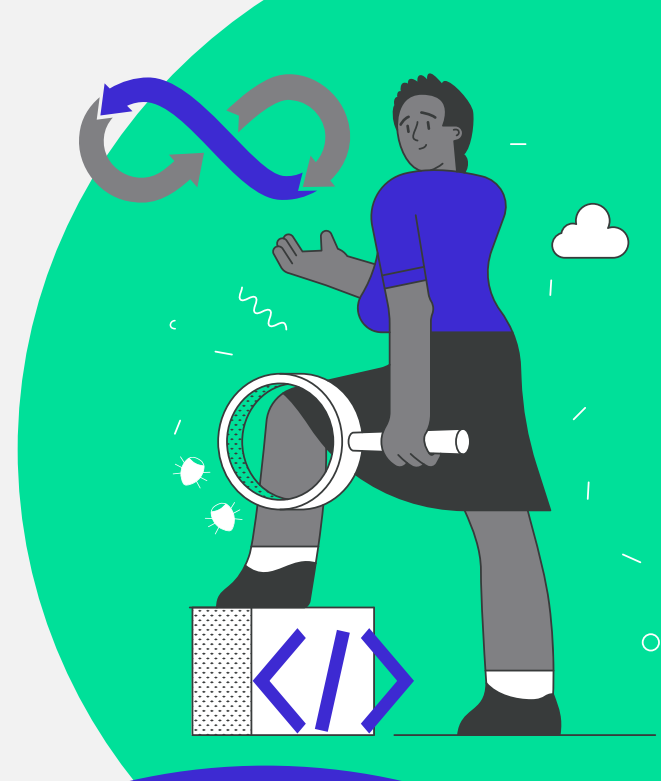
# Open Source Maintainers

Exploring the people, practices,  
and constraints facing the  
world's most critical open  
source software projects

July 2023

Alex Salkever, *Linux Foundation Research*

Foreword by Shuah Khan, Fellow, *The Linux Foundation*



# Open Source Maintainers

75% of interviewees serve both as project maintainers and contributors of code.



Less than 1/3 of interviewees say their project has a formal DEI program.



38% of interviewees say that they feel a high degree of support from their employers for their open source work.



34% of interviewees say their project has a formal mentorship program.

Concerns about code vulnerabilities must be balanced against critical operational considerations, such as recruiting new contributors.



53% of interviewees indicate their project has a formal new contributor recruitment process.



Only 35% of interviewees say their project has a strong new contributor pipeline.

62% of interviewees are employed to work full-time on their projects.

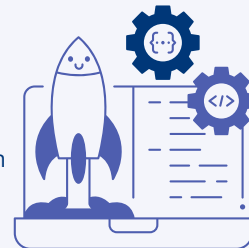


39% of interviewees feel that open source software work is highly valued at their organization.



Interviewees recognize that specific behaviors discourage new contributors, such as not responding to pull requests in a timely manner.

**Embrace automation.** While it does not replace the human touch, it does provide key benefits such as scale and instant response times.



A mention or shout-out on social media or a community call can encourage new contributors to tackle hard problems.



# Contents

- Foreword .....4
- Executive Summary .....5
- Introduction .....7
- Methodology .....8
- Observations on the demographics of maintainers and contributors.....9
  - Career path of maintainers.....9
  - Contributor, maintainer, or both? .....9
  - Working full time or part time on OSS projects.....10
  - Super contributor and maintainer project onboarding experience .....11
  - Value maintainers receive from open source maintainership .....13
- Maintainer best practices.....14
  - Contributor experience.....14
  - Community governance and management .....19
  - Documentation .....21
  - Funding and other forms of support.....23
  - Diversity .....25
  - Preventing maintainer burnout .....26
- Conclusion.....28
- Determine your project attributes.....28
- Create a strategy and road map based on your attributes.....28
- Identify key metrics for your project, and track them regularly.....29
- Identify best practices for your project. ....29
- Acknowledgments.....30
- About the Author .....30

## Foreword

Open source software is the backbone of our world infrastructure. From financial services and healthcare, to telecommunications and the internet—it is no secret that open source runs the world..

It powers smartphones and laptops, communications and data infrastructure, our logistics systems, and our government. From Linux and MySQL to thousands of viable projects in everything from front-end frameworks to databases to container orchestration and telemetry, open source software is everywhere. We all know code doesn't write itself. At the core of every successful open source project are the maintainers and primary contributors.

From designing and writing the first lines of code to overseeing and nurturing massive open source communities such as Linux and Kubernetes, the maintainers and contributors drive innovation, progress, and healthy ecosystems. They set the rules and the tone—or set up the process that empowers healthy self-governance and continuity of their communities. Without maintainers and contributors, open source software would quickly become outdated and insecure. For critical software projects, the burden is greater.

A relatively small number of open source projects comprise an astonishingly high percentage of software in use today. It is imperative that we understand more about these critical projects and capture the lessons of those who run them. Their lessons can inform future founders and maintainers of critical projects. And, by making open source broadly more sustainable, we believe we can increase the pool of successful open source projects and make maintainership more fulfilling and less taxing.

Open source always has been and will be about the people who take the risk and make the projects happen. As a foundation dedicated to the success of open source, we owe it to the maintainers to make their lives better and to illuminate a better path forward. We hope this report can contribute to this effort by recording the wisdom and learnings of some of the most successful maintainers from some of the most critical projects and making them available to all.

Sincerely,

**Shuah Khan**  
**LF Fellow / Maintainer**  
**The Linux Project**

# Executive Summary

From operating systems to databases to programming languages, the world increasingly relies on open source software (OSS). It has become the foundation of much of the global technology infrastructure. Unlike proprietary software, which engineers construct when working in a closed environment for a firm explicitly seeking profits from code, OSS relies on a web of maintainers who work, often without pay, in initial project stages to build software. Research has found that our technology infrastructure relies heavily on a few hundred open source projects. These projects represent a disproportionate percentage of software dependencies. The maintainers of these projects bear a tremendous burden. Projects they oversee and steward are responsible for much of the global economy. Disruptions to these projects can cause massive problems and outages.

In some cases, these maintainers work alone, with little or no organizational or financial support for their projects. Even when maintainers enjoy significant organizational and financial support, including explicit duties as part of their salaries, the job of maintaining the most important OSS—the job of super maintainers—remains challenging to quantify and illuminate. This report seeks to document how maintainers become maintainers, their experiences and observations for growing successful OSS projects, and their tools and best practices for balancing requirements to grow a software community and live a fulfilling and sufficiently remunerative life.

## Methodology

For this report, LF Research conducted detailed qualitative interviews with 32 super maintainers from projects our research identified as among the top 200 critical OSS projects. The maintainers came from a wide variety of employment histories but shared many sentiments and common experiences.

As projects have grown in scale and complexity, maintainers face increased demands on their time. This led to less effort to welcome new contributors. Early contributors benefited from less competition and more opportunities to engage with project founders. Despite concerted efforts to maintain a positive contributor experience, maintainers fear it has deteriorated over time. They consistently highlighted the imperative for continued focus on fostering a supportive environment for new contributors. This is especially important for mature projects where attracting new maintainers is more challenging and for parts of projects that require more technical acumen (memory management, to name one example).

As observed in multiple previous studies by LF Research, maintainers enjoyed the intrinsic rewards of working on open source, including a sense of community and camaraderie, work on cutting-edge technology, the ability to set their own course and prioritize their activities, and the sense of achievement in watching a community take shape. Maintainers recognized that they benefitted

extrinsically in career trajectory, stature, and respect in the community and career confidence that another job would also be available. Many of the maintainers received their current employment due to their community work. In some instances, companies hired maintainers specifically because of their expertise and ability to influence project dynamics. However, most maintainers expressed concern that their own organizations did not sufficiently recognize their efforts.

## Growing contributions

Maintainers interviewed collectively supplied a wide range of best practices and wisdom for fostering and growing communities, including prioritizing and making time for personal contributor engagement; using inclusive language; providing multiple communication channels for engagement; and providing clear onboarding resources. Successful projects further supported first-time contributors by suggesting suitable bugs or pull requests (PRs), either through flags in GitHub or in response to questions floated in community channels. Maintainers should always seek to identify those with higher capabilities or levels of persistence and nurture them to build the next generation of super maintainers and project leads. Establishing triage processes for new PRs and organizing team efforts to handle commits and PRs in a timely fashion further enhances the contributor experience.

# Executive Summary

## Governance and control

All interviewees felt that community governance and management are crucial for the long-term success of a project but are often overlooked in the early stages. Best practices include establishing a code of conduct and promoting civility both explicitly and implicitly (through actions and the tone of project leadership). Distributing power can prevent decision-making bottlenecks that frustrate contributors and slow down community innovation. Neutrality in community management ensures fair treatment of all contributors and encourages developer participation from multiple organizations.

## Documentation

Not surprisingly, maintainers expressed concern that their project needed to improve documentation. Best practices suggested by maintainers included making sure that project leads demonstrate documentation is a first-class citizen in a project with comparable recognition to code contributions; hiring a documentation coordinator; requiring documentation to be submitted with each code contribution; and creating formal events or processes around documentation to make it easier to contribute.

## Funding

Earning enough money to live on was a concern for only one of the interviewees (who also worked on an unaffiliated smaller project in the JavaScript community). That said, multiple maintainers expressed frustration that critical open source projects they maintained or knew about languished or did not ship new versions because of the lack of funding support. (Many maintainers interviewed maintain multiple projects, including some that are not part of their employment responsibilities.) Insufficient funding mechanisms for smaller and mid-sized critical projects were a key driver in the decisions of maintainers to seek out full-time employment at an organization willing to underwrite their work. Of the handful of independent maintainers interviewed, all expressed concerns about how to sustain open source projects that do not have foundations or large corporate funders.

## Diversity

Most maintainers struggled to generate a sufficiently diverse set of contributors and project leads. Roughly half of the maintainers had no explicit diversity efforts or goals. Only a handful participated in diversity programs such as Outreachy. Maintainers that successfully fostered diversity made DEI goals top-level project goals with associated governance efforts and participated in diversity programs. Despite best intentions, open source has a long way to go regarding diversity.

## Preventing burnout

Maintainers discussed various strategies and practices that open source maintainers use to prevent burnout. These practices include recognizing that open source work is never finished, designing a lifestyle that balances work and personal pursuits, avoiding taking on unpaid projects that require excessive administrative work, automating workflows to increase efficiency, setting boundaries with regard to communication and work hours, and taking breaks when feeling burnt out. Many of these strategies involve cultivating self-awareness and being realistic about personal limitations to prevent burnout while still contributing to open source projects.

# Introduction

The health of the OSS ecosystem depends on the quality and success of a core group of the most active and responsible maintainers and contributors. The open source ecosystem and, by extension, the technology infrastructure of the world is heavily dependent on a relatively small number of projects numbering in the hundreds. LF Research has studied and attempted to categorize the most critical projects in terms of dependencies in ongoing work with Harvard University in our Census II program (most recently with [“Vulnerabilities in the Core, a Preliminary Report and Census II of Open Source Software”](#) and [“Census II of Free and Open Source Software—Application Libraries.”](#)) Amidst the millions of open source projects, a critical group of projects and their maintainers and contributors occupy an outsized role. LF Research calls these leaders of the open source realm “super maintainers” and “super contributors,” who collectively form a group of “super coders.”

The burdens on them are heavy. Millions of users and systems around the world download the projects they oversee. Security vulnerabilities in their projects may cause massive global disruption. Vulnerabilities in their code can “break the Internet.” They also must balance these concerns against more mundane but equally critical operational considerations, such as the need

to constantly recruit new contributors, set up proper systems of governance and adjudication for inter-project disputes, and achieve greater levels of diversity, all the while ensuring that their project continues to innovate and iterate with new technology cycles.

This research is the result of interviews with more than 30 maintainers of, and core contributors to, the world's most critical OSS projects, many of which were among the most widely used application libraries identified in the Census II report published by the Linux Foundation in collaboration with the Laboratory of Innovation Science at Harvard University.

Together, the interviews capture the insight and wisdom of the people at the helm of some of the most critical open source projects. Interviewees describe how to effectively start, scale, manage, and innovate within open source projects and identify opportunities for open source communities and enterprises alike to better support them in their important work.

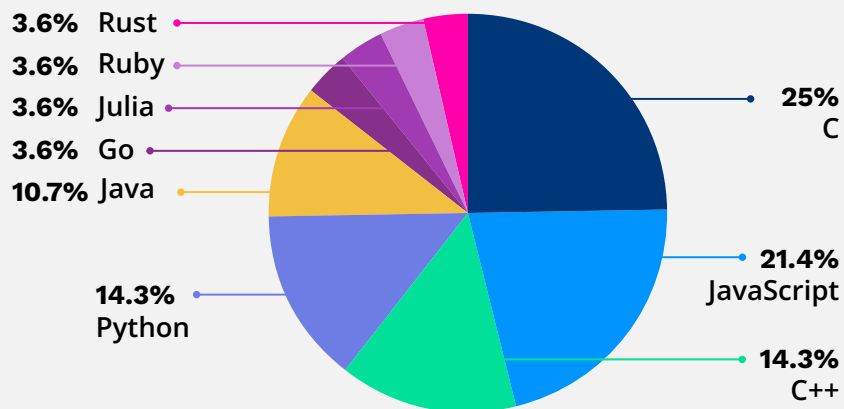
# Methodology

LF Research conducted 32 interviews of roughly one hour with super coders from critical projects. Interviewees for this study were contributors to, or maintainers of, a diverse array of software that occupies different roles and niches in the vast open source ecosystem. This includes front-end libraries (Babel, Webpack, React, Storybook); operating systems (Linux); infrastructure (containerd, Kubernetes); package repositories and managers (Rust Cargo, npm, RubyGems, Gradle, Apache Maven / Maven Central); databases and storage (PostgreSQL, Ceph); developer tooling (LLVM, Git, cURL); DevOps (Salt); lower level languages (Julia); and data analysis and machine learning frameworks and applications (PyTorch, NumPy, Jupyter).

LF Research intentionally included projects at various stages of maturity and age, from a few years to multiple decades, and in size from a single maintainer to projects with thousands of contributors and dozens of maintainers. Lastly, LF Research included projects using multiple languages, including C/C++, Go, Rust, JavaScript, Java, Node.js, Python, Ruby, Julia, and more. In doing so, LF Research identified common patterns, challenges, and best practices based on these three core parameters.

There are millions of OSS programs available for free download, hosted in multiple locations, including on the leading version control and programming collaboration tools (GitHub, GitLab, Bitbucket) as well as on web servers and in other locations. The data in the Census report provides a limited but important view into the most widely used OSS. The software projects assessed by Census II had various organizational structures. Some were single maintainers with no foundation or funding support. Some projects were larger and more complex software projects, with multiple committees and bodies as part of project governance. A single company wholly controlled some projects, while a large community of contributors wrote others. The aggregated data measured the dependency graph and assessed which software packages the largest group of end users used and was dependent upon. The data was anonymized and modified to prevent any linkage to the organizations running the applications themselves. In some instances, usage data (primarily downloads per month) was also studied and available through other data sources that collect and analyze data on OSS usage, such as Libraries.io. Libraries.io collects usage data from package repositories and managers, a key nexus of open source dependency data and application usage.

**FIGURE 1**  
**LANGUAGES OF PROJECTS SURVEYED**





# Observations on the demographics of maintainers and contributors

The 32 interviews included maintainers of various ages and from a wide variety of locations. Maintainers hailed from a half dozen countries. Six of the interviewees were female, and the remainder were male. All were developers and engineers with considerable experience. They worked for companies ranging from large multinationals to small consultancies. One worked solely on their open source project. Most were working either at very large or small companies. Large companies represented included Red Hat, Amazon, Microsoft, IBM, Meta, and VMware. Few were working at “mid-sized companies.” Two of the interviewees worked at the Linux Foundation. Three of the interviews focused on previous project work; those interviewees had moved on from maintainer and contributor roles, but their experience remains useful and valuable.

## Career path of maintainers

Maintainers of and contributors to these critical projects came from a variety of academic backgrounds. A significant percentage studied computer science or software engineering in university as a major or a minor; several worked on the software they would later become a maintainer of while in school as an undergraduate or graduate student.

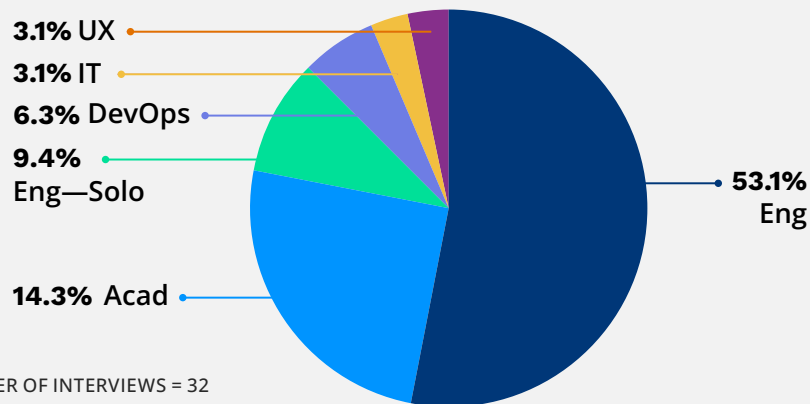
This was particularly true for maintainers and contributors working on lower-level and more complex projects such as the Linux kernel and distributions, Git, and databases. Those maintainers also were more likely to take jobs out of school at large established software companies, where they took part in open source work as part of a lab or part of a growing open source practice.

Many of these maintainers initially started in open source through exposure to and work with the Linux operating system. There was also a correlation between the language of the project and the percentage of contributors who studied computer science; projects written in C-type languages tended to attract university-trained computer scientists more than projects written in other languages. Another significant group of maintainers was academics who began to work on open source to better solve their own computing problems. Only one of the maintainers LF Research interviewed with an academic background remains primarily employed as an academic; most migrate to a role at a technology company or foundation once their project gathers sufficient critical mass.

## Contributor, maintainer, or both?

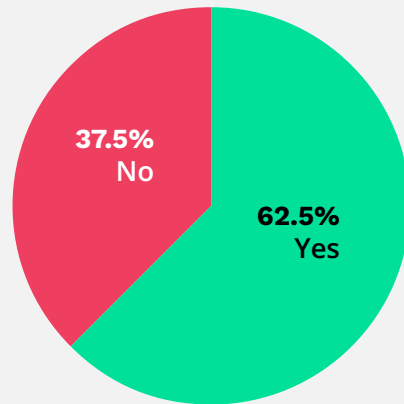
LF Research defines a contributor as someone who writes and submits code to a project and a maintainer as someone who works in the management of a project, including code review, triage,

**FIGURE 2**  
**PRIOR CAREER ROLES OF MAINTAINERS**

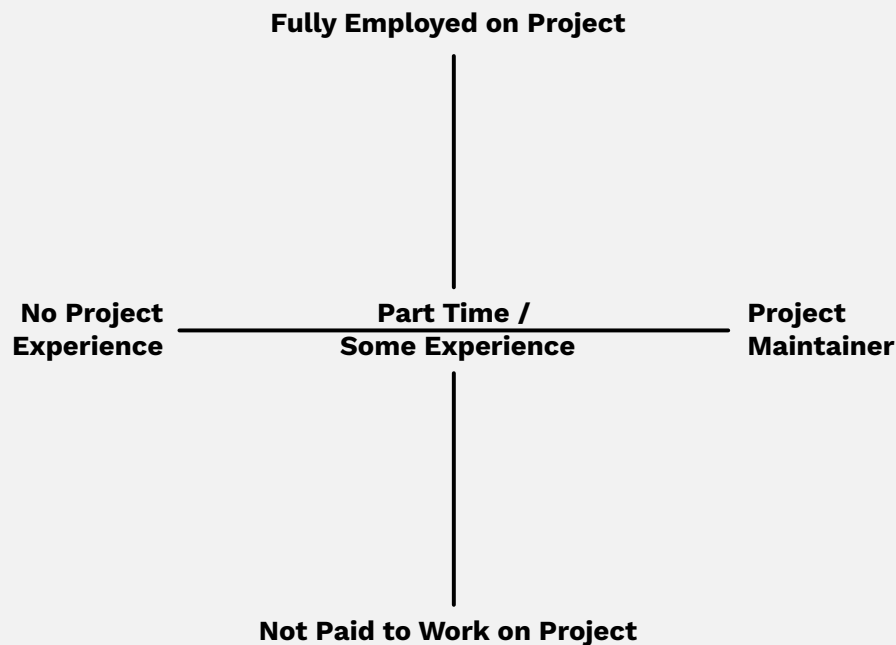


NUMBER OF INTERVIEWS = 32

**FIGURE 3  
MAINTAINERS WORKING  
FTE ON OSS PROJECTS**



**FIGURE 4  
MAINTAINER ARCHETYPE GRID**



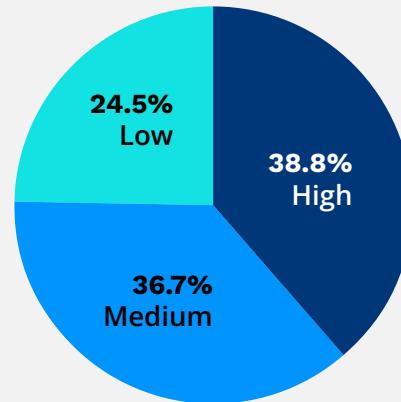
test, security, build and infrastructure, and release management. A total of 75% of interviewees served both as project maintainers and contributors of code, with 25% serving only as maintainers. None were only contributing code, as one would expect when interviewing maintainers. The percentage of their time spent working on open source projects varied widely, averaging 70% of their time. Our interviews covered only one project operating with just one maintainer; every other project examined had more than one maintainer helping with the workload.

Over the course of their time working on their project, most of the interviewees followed a progression path from contributor to core contributor to maintainer. Inside the larger organizations, maintainers tended to juggle multiple roles and tasks. For example, Brian Granger, a Project Jupyter maintainer, initially wrote much of the project code. Today he focuses on fundraising, management, technical architecture design, reviewing code, and improving the UX practice for Jupyter. Granger, who works at Amazon Web Services, also continues to be involved in the community governance of Jupyter as a member of its Executive Council. Shuah Khan at the Linux Foundation oversees test and QA processes for the Linux kernel. She also works to improve project documentation and is actively mentoring 13 contributors as part of her efforts to increase project diversity. Khan initially started out primarily contributing expertise and architectural design to the test and QA infrastructure of the Linux kernel but then progressed to roles that included more managerial activities.

### Working full time or part time on OSS projects

All but two of the interviewees had full-time employment with a company that supported investing their time in the project. Another significant percentage worked at venture-backed companies of significant scale, including Vercel, Chromatic, and Oxide Computer. Only one maintainer received support solely from sponsorships, donors, and other forms of ad hoc project

**FIGURE 5**  
**PERCEIVED DEGREE**  
**OF ORGANIZATIONAL**  
**SUPPORT**



“

*“If I step back from the job I have and from everything else and just ask, ‘Okay, what could I do that would have the greatest impact on society, technology, humankind?’ as a whole, I’d be hard-pressed to find something better than working on open source.”*

—BRIAN GRANGER, CO-CREATOR AND LEAD MAINTAINER, JUPYTER

funding. Another was in between jobs but had been employed steadily working in open source for the past decade, primarily as an open source evangelist.

The relationship between the interviewee, their employer, and their project was varied and often fluid. In some instances, the employee was hired full time to come work on a project without having worked on the project specifically before. In some instances, an employee was hired specifically because of their position in the community and their ability to help the company understand the project road map and support upstream contributions.

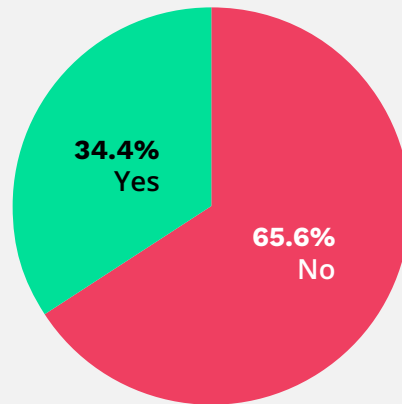
In some instances, an employee was hired with the understanding that they would spend part of their time working on a project that is strategic to their employer. In a handful of the cases covered here, interviewees worked for a company that employed all the primary maintainers of a project. In a rare set of instances, the interviewee was fully employed, but the employer was indifferent to their work on the project. In two instances, an interviewee worked for a company that specialized in helping customers by crafting features for a specific project.

## **Super contributor and maintainer project onboarding experience**

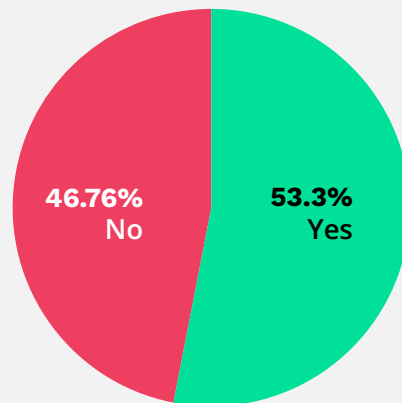
With few exceptions, interviewees enjoyed a welcoming onboarding experience when they first became involved in the project. This is not to say in every instance that it was necessarily easy or obvious to find a set of first issues to work on. However, universally, interviewees who were not original founders reported a positive first experience entering the community.

Nevertheless, multiple interviewees who had been contributors or maintainers since or near the inception of their projects said that they doubted their community today would be as welcoming due to the scale of the mature community, the present demands on maintainers, and the increased complexity of the project code

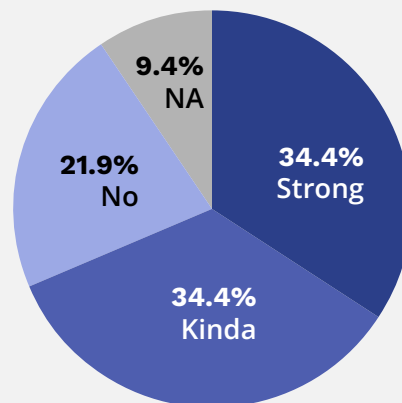
**FIGURE 6**  
**PROJECT HAS FORMAL MENTORSHIP PROGRAM**



**FIGURE 7**  
**PROJECT HAS FORMAL NEW CONTRIBUTOR RECRUITMENT PROCESS / PLAN**



**FIGURE 8**  
**PERCEIVED STRENGTH OF NEW CONTRIBUTOR PIPELINE**



and architecture. When he first contributed code to PostgreSQL, Andres Freund said, "... because the responses were so quick and in-depth, it was like, 'Cool! That's way more than review or whatever I've ever gotten in my work life.' I don't think that's the case anymore these days because just the volume has increased so much that that kind of quick response and in-depth response is not the norm anymore."

Several interviewees remarked that their initial contributions, often error-riddled, would have less likely received a warm response in the project today. Their same contributions, if submitted today, would require considerable effort to push their PRs over the line and get them merged. In addition, early contributors felt they were able to tackle more substantive problems since they were so early in a project. Competition for maintainers' time was less stiff, so rich engagements with project founders (via email or in GitHub) were more likely and sustainable. Also, there were many more areas where contributions were welcomed because the project community was less mature, and expertise in key areas of project development remained lightly covered. In at least one instance, a maintainer had carefully observed community behavior and picked an area of contribution that was extremely complex and had few participants precisely to stand out and improve their chances of acceptance into the community.

The maintainers interviewed recognize specific behaviors that discourage new contributors:

- Maintainers have been unable to respond to PRs in a timely manner.
- Maintainers stopped spending extended time helping new contributors debug PRs.
- Maintainers stopped actively triaging new contributors to identify potentially valuable talent.
- Contributors struggled to find an entry point for conversations and input.

“Because just the volume (of work and PRs) has increased so much that that kind of quick response and in-depth response (which I received) is not the norm anymore, at least in the PostgreSQL community, I don't even know whether I could succeed today or whether I would be hooked again like I was when I started out,” explains Freund, a member of the core committer team of the project. In other words, contributor experience tended to degrade, in their opinion, over time. This was often despite concerted efforts to address and maintain contributor experience.

## Value maintainers receive from open source maintainership

Maintainers universally expressed gratitude for open source and their ability to work in a system with open source values. Specific benefits derived by maintainers for their activities include both intrinsic and extrinsic values. The intrinsic values LF Research has commonly seen expressed in previous surveys of open source maintainers and contributors, such as the [FOSS Contributor Survey Report](#), include:

- A sense of community and camaraderie
- The ability to work on cutting-edge technology
- The ability to set their own course and prioritize their activities, the ability to work with a global community of peers, and the joy of creating something from scratch
- A sense of achievement in watching a community take shape and flight
- The chance to work with really smart people

“When I first met some of the people in person in the community, I was giving everybody hugs. It was a very emotional moment actually because we had sort of been so invested and committed for so many years working on things together,” says one of the lead maintainers of Julia.

*“Open source, it is a life-changing experience for me. Coming from a closed-source world, I feel like I am in the driver's seat, I would say, as opposed to riding as a passenger in the backseat. I feel like I can control my career; I have direct control over what I do and what my contributions are.”*

—SHUAH KHAN, MAINTAINER, LINUX KERNEL



Extrinsic values included access to better employment; the ability to work for a variety of employers and choose their mode, location, and style of work; and external status signals such as association with a noted or trending project. In addition, several maintainers mentioned that their open source work, while not directly contributing to their day-to-day work, did contribute to their attractiveness to employers. Only one of the interviewees expressed financial concerns over working in open source as a career choice. Not coincidentally, that maintainer was heading a project with no clear commercial entity or monetary engine, one entirely dependent on donations and sponsorships. The majority of the maintainers expressing funding concerns for their projects were working on JavaScript projects, an area that has traditionally struggled to obtain sufficient funding due to the dynamics of the JavaScript ecosystem and low levels of corporate funding for independently maintained JavaScript projects.

# Maintainer best practices

As part of each interview, LF Research asked maintainers and contributors what were the best practices that contributed to the success of their project. While community health is an inexact science, most of the communities covered in this research effort were healthy by the most basic metric—they were shipping new code and versions.

Several maintainers voiced concerns that their community was struggling to find fresh contributors and that this challenge could negatively impact the project down the road. Maintainers all expressed strong views on how to maintain community health. The size, language, and life cycle stage of the project impacted these views.




How each project's best practices came about varied widely. Most maintainers created some best practices from personal hands-on experience and through trial and error. Others were codified in the community as specific parts of governance and rules. Many projects had gone through periods of change where a new set of best practices was adopted to enable smoother community interaction, execution, and growth. In hindsight, the projects treated their own practices as they did code, subjecting them to constant scrutiny and review for potential ways to improve.

There is no one set of comprehensive best practices that will work in all cases. However, maintainers and core contributors demonstrated great creativity in building out personalized and organizational best practices that met their needs. Here is a breakdown of the best practices identified as useful and helpful by interviewees. This list is grouped into specific areas of practice, although in many cases, a best practice affects multiple areas, including contributor experience; community governance; documentation; fundraising and in-kind contribution generation; diversity; and burnout prevention. In the sections below, LF Research expands on each of them.

## Contributor experience

Contributors are the lifeblood of neutral open source projects. Many of the successful maintainers prioritized contributor experience very early in the project, working to encourage contributors to comment, file bugs, and, ultimately, submit suggestions for improvements to code (in the form of PRs). Some maintainers undertook heroic measures to encourage contributors. For example, Norbert de Langen, the lead maintainer of the Storybook project, sent a meeting scheduling link with a request to speak in person to anyone who sent him an email with a suggestion or question about the project code. De Langen met with over 200 people using this method during the first year of the project. He estimates that nearly 20% of those he met with later became repeat contributors to Storybook.



Maintainers that build successful projects often go to extensive lengths to help first-time contributors improve their contribution and learn how to prepare a code submission, such as checking the format and syntax of code using automated check systems. Gareth Greenway, currently a maintainer of the Salt project, had submitted an error-riddled PR that the lead project creator, Thomas Hatch, read. The original creator gave Greenway positive feedback and worked with him to fix the code. Greenway went on to submit numerous other PRs and later became a Salt maintainer.

Here is a list of some other common ways that successful maintainers build a strong contributor experience.

#### Respond personally to first-time contributors.

Personally respond to first-time contributors when they start to engage with the community. This is more viable in early-stage projects with smaller communities, but many maintainers still attempt to respond personally consistently. For smaller, more contained projects, personal response time is less of a challenge. For example, in Linux toolchains or in projects such as containerd, which do not require large numbers of contributors and are written in more challenging software languages such as C++, maintainers do not see a high volume of inbound messages from potential contributors or maintainers. (Note: This also indicates challenges in recruiting new community members and contributors, particularly for older projects at the later stages of their life cycle).

#### Set up automated greeting bots and an onboarding guide.

Automation does not replace the human touch, but it does provide key benefits such as easy scale and instant response times. Particularly for mid-sized and larger communities, maintainers generally set up an automated greeting bot to greet new community members in Slack or Discord and guide them toward onboarding resources. Linkerd, Kubernetes, Ansible, GitHub, and many other organizations use this capability. Some

organizations use bots in GitHub to greet new contributors. "Jupyter Lab (part of the Project Jupyter) uses a GitHub bot to welcome new contributors. If we detect you opening an issue or pull request, we respond, 'Hey, this is your first time contributing. Welcome to the project. Here's how you can participate in the community,'" explains Granger.

#### Use inclusive language.

This can meaningfully improve participation for diverse groups of current and prospective contributors. Inclusive language is neutral with respect to race or gender but also considers regional language differences and the ability of non-English speakers to easily understand project communications. Inclusive language can be welcoming, engaging, and energetic. This is not to say a project must communicate solely in an anodyne style but that project leaders should be cognizant of the communications requirements for contributors and other project participants. For example, a project should seek to minimize regional expressions, such as sports references and local jokes, to simplify communications. There are a variety of automated analysis tools that can provide inclusive language suggestions for emails, Slack, and online documents and pages. Grammarly, the most popular online grammar-checking tool, now offers [suggestions for inclusive and gender-neutral language](#). [Textio](#) is a popular online tool that screens job listing language for both explicit and implicit bias and suggests language that is both explicitly and implicitly more inclusive. Common sense also can go a long way toward making language more inclusive.



Create a simple page that aggregates all the different entry points and communications channels for a project.

Many projects have multiple means of communication, including email lists, chat (Slack, Discord, IRC), GitHub repos, project channels on YouTube, Wikis, and more. A simple way to help new contributors connect with the community is to make it easy to find all these channels, as Ceph does with their [“get involved” page](#).

Create an onboarding or “how to contribute” section in the project documentation or a markdown file on the project’s GitHub page.

This includes an explanation of how the community works, links to tutorials and key documentation, and ideally identifies ways to contribute and participate along with ways to connect with key community leaders and members. Almost every project does have an onboarding page somewhere in its documentation and GitHub. The best practice is to link to them from multiple locations to ensure new contributors can easily find them. The more information on this page, the better. For example, [Jupyter’s CONTRIBUTING.MD](#) page on GitHub contains granular detail down to links to how to do regression tests for any contributions.

Suggest bugs or PRs for first-time contributors (often called “good first issue” bugs or PRs), or identify bugs that could use help.

In many projects, contributors are hesitant to dive in and submit a PR because they do not know whether a bug is “owned” by anyone in the project. (For example, someone who controls a module may have a trusted cadre of bug fixers.) This is where labels on bugs for new contributors come in handy. These first-time bugs can be a creation of docs or more basic code fixes, but it really depends on the preferences of project leadership. Many projects now have a label for these items in their GitHub repos. For example, VSCode has a [“help](#)

*“A maintainer’s job is being responsive to the community and contributors, trying to understand where they’re coming from and giving them feedback. A lot of times, some of our major bugs have been caught or addressed in time because we’ve been responsive.”*

—NEHA OJHA, Ceph

[wanted](#)” label to identify bugs that are unclaimed. Storybook.js has a [“good first issue”](#) label that steers contributors to issues that may be less complex. A key part of using these labels is ensuring that they always have a backlog. Storybook, for example, at the time of this writing, has 26 issues under the label. For contributors that do not know where to start, this is an ideal method to steer them.

Establish criteria or a mechanism to identify contributors with higher-than-average capabilities.

While all contributors may be valuable, those with suitable skills may be more valuable because of their comparatively shorter learning curves. For example, in projects written in C-type languages, fluency in one C language is a huge benefit. Contributors that demonstrate technical acumen through a high-quality PR or even sheer persistence in tackling harder problems can be encouraged to take on more complex initial projects or connected with mentors working on more challenging aspects of a project code base or sub-systems, such as memory management, compilers, or networking. Maintainers identified technical acumen primarily by noting code quality or thinking of an initial PR or in the content of email conversations with a potential contributor about the project.



Both underscore the importance of outreach and direct communication with new contributors, particularly in projects where it is hard to find contributors to tackle hard problems and project tasks (memory management, compilers, and security, to name a few examples). Some projects require knowledge overlaps that can act as a filter for contributors with potential. For example, NumPy is primarily written by and for scientists, engineers, and people using Python for data analysis, but the code for the project is in C. The NumPy maintainers know that if any scientists who have C coding skills ask to contribute, they are likely to be strong contributors.

**Rather than reject flawed contributions, offer suggestions on how to improve and start a dialogue.**

For some maintainers, a potential contributor who communicates clearly and logically in their PR or via email passes the sniff test. "Better than the patch quality was people who submitted something and explained what they were trying to do. I came back to them and said, 'I see what you're trying to do, but X, Y, Z,' and you had a good conversation with them," explains Jeff King, a former lead maintainer of Git.

**Reward efforts to tackle the hardest problems.**

Hard problems can become "showstopper" risks and major bottlenecks for projects. More challenging engineering tasks generally attract fewer aspirants. This problem tends to grow as projects mature and institutional knowledge accumulates in a small cadre of experts. For this reason, project maintainers indicated they made extraordinary efforts to nurture contributors who show interest in solving or working on harder problems, even if they are project newcomers. There are several simple yet meaningful ways to acknowledge and reward these efforts. A mention in release notes, a shout-out in social media, and a mention on a community call are just a few of the zero-cost, low-effort ways to encourage efforts to tackle hard problems. Maintainers may want to go further. This could mean sending contributors working on hard problems special swag, helping them to author blog posts in

public settings and on the project site, and assisting them in lining up conference presentations about their work, to name a few examples. Above all, maintainers should work especially hard to nurture these contributors by ensuring timely responses to questions, comments, emails, or code submissions. This demonstrates interest in the scarcest of maintainer commodities—their time.

**Strive to lower the bar for contributions.**

Some contributors may wish to participate but get frustrated due to a steep learning curve, lack of documentation, or challenges in using the technology. This can scare off contributors who may ultimately become quite valuable and productive. Smart maintainers recognize this problem and set out to address it as part of their core work.

At Ceph, Neha Ojha has long struggled against the perception that the project is hard to use and get started on. To combat this, one of her top priorities is "... lowering the bar. We can always lower the bar for new contributors and new users," says Ojha. For her, some of that emphasis has come on documentation. "I emphasize improving the Ceph documentation because I felt like, if we have the right documentation in place, the first battle is already won. Then users can at least get Ceph up and running, and we have those small wins to keep them engaged."

**Create happy milestones to encourage new contributors who are showing promise and dedication.**

For example, Storybook quietly adds a contributor to the project team in GitHub after their second PR is merged. The contributor is notified and is generally delighted. (This also unlocks other GitHub benefits for them, such as a free Copilot account.) "People are surprised, and it makes them really happy," says de Langen of Storybook. Other maintainers suggest a simple means of recognition, such as sending out a special limited-edition release sticker to all contributors cited on each project release. NumPy adds a "+" sign next to first-time contributors' names to show special gratitude in release notes emails.

*“The very first pull request I submitted had close to 500 lint errors. I think I still hold the record for the most lint errors of any contribution to Salt. A lot of open source projects would just have told me, ‘This is garbage. Take a hike.’ Instead, Tom Hatch (the creator of the Salt project) said, ‘This is great! We have needed this for a long time. It looks like you’ve got some lint errors. Let’s get those fixed and get this merged.’”*

—GARETH GREENAWAY, SALT

#### Set up office hours specifically for new contributors.

This time can be used to help guide them through the contribution process and raise the probability their contribution will be quickly merged. New contributors also often enjoy mentorship from and contact with maintainers. Shuah Khan of the Linux kernel mentors multiple contributors of diverse backgrounds by meeting with them periodically, answering their questions via email, and helping them prepare contributions. This is part of her job in the project. “They will be submitting patches that we need to consider upstream, so it is better if their work is mentored to give them a better understanding of how to land a patch,” explains Khan. “It benefits the project overall by improving patch quality from new contributors.”

#### Establish and follow triage processes for new PRs.

This may seem obvious. Still, the key is to go beyond setting up the process and to give ownership of triage to a key project member or group of members (this can mean rotating members through stints as the triage master). Most projects already aspire to responsible triage of new patches and requests, but often work and life can get in the way. Without a system to handle triage and support from project leadership to prioritize it, triage tends to fall to the bottom of priority lists. Unless triage and first-time response is aggressively prioritized, it can become an ad hoc / best efforts approach. This may turn off contributors because it does not manage expectations and may result in lengthy delays. For example, the Salt maintainers overhauled their triage process and created specific roles and coverage to ensure triage moved along and followed the same process. “Every new issue that comes in goes through proper triage these days,” says Pedro Algarvio, a Salt maintainer, noting this was not always the case.

#### Don’t be afraid to say no.

New contributors may propose ideas and submit patches that could have significant impacts on the workings of project code and subsystems with cascading and unintended consequences. Strong maintainers strive to respond quickly and explain politely but firmly why their idea or patch may be problematic and that it is unlikely to be accepted. In addition, the response (which should remain public) can help guide later contributors or serve as an easy way to explain the point to other new contributors in the future. “No is temporary, but yes is permanent. Be very careful about what code you agree to add because it has long-term consequences,” says Laura Abbott, who was one of three principal kernel engineers of the Fedora Project (a downstream Linux distribution) at Red Hat and was a Linux maintainer. Beyond kernel engineering, Abbott was highly active in the Fedora and broader Linux community, both as a maintainer and as a mentor of contributors and wrangler of community concerns. Explains Abbott further, “Listen to what people are asking, but also just be aware that it can be a lot harder to remove code than add code, so make sure that it’s added with this in mind.”

#### Create team efforts to “burst” handle commits and PRs.

The PostgreSQL team regularly organizes “committfests,” where every meaningful pending code submission and patch receives a response from the project team. This system works best if there is a cadence and the activities are repeated several times a year.

#### Ensure that the right tone is set at the top.

The founding and leadership DNA of open source projects is critical to further success and community growth. The best project founders and leaders facilitate and support processes and efforts to improve governance and grow contributorship. The tone of inclusion and community health must be set at the top—usually by the project founder and the small team that starts the project.

## Community governance and management

Community governance and management are often left out of consideration in the early stages of a project. This is not surprising. In the early stages, a project creator is focused on getting to a minimum viable project and creating a strong software foundation. When a project is birthed inside an organization, then governance is often an internal function and focused on internal needs rather than the longer-term requirements of building a sustainable community. Any complex project eventually requires codified governance and a management structure—even if there is a natural “lead maintainer” overseeing the project (e.g., the creator).

For small projects, explicit governance policies and structure may not be necessary for day-to-day management but become essential for adjudicating disputes and for making long-term direction decisions about a project involving the community. Codified governance is often overlooked when things are going great but suddenly becomes critical when the community hits an issue. Here is a collection of some of the best practices for community governance and management provided by the interviewees. Some of these are obvious—such as establishing a code of conduct. Others are less obvious, more idiomatic, and may not work broadly across all projects but have been successful in their specific project.

### Establish a code of conduct.

This is now common practice for successful open source projects that wish to maintain a welcoming environment. Examples of codes of conduct are easy to find. Rarely do open source project leadership teams and project members object to codes of conduct. However, they are critical to prevent conflicts and to set a minimum standard for cordial interactions. In addition, it is often critical to document the guidelines and processes for dealing with reported conduct violations, removing bias from decisions, and dealing with repeat offenders.

*“I tend to view the changes and ignore who’s actually behind it. Even if there’s a regular contributor doing this commit (or proposing a change) or a complete newbie, I try to view both identically.”*

—DANIEL STENBERG, LEAD MAINTAINER, CURL

“

### Establish prevailing community norms of civility.

Codes of conduct are necessary but are not sufficient on their own for setting the right tone. Many of the interviewees consistently mentioned that part of why they continued to contribute to a community—and later agreed to become a maintainer—is the civility of the community. Experienced maintainers also said that they politely requested community members to soften their tone if it appeared that a discussion was getting heated. This was a way to make a community conversation more welcoming by visibly encouraging abrasive but well-meaning community members to use a less abrasive tone in dialogue.

### Design yourself out of your job ASAP.

Maintainers discussed how challenging it was to relinquish power and control over their project emotionally. Most felt a strong sense of attachment to their projects. However, they recognized the importance of giving up control to incentivize contributors and allow them to create and build a more self-sustaining and healthy community. A handful of maintainers aggressively sought to give responsibility out to any developers or volunteers who expressed

interest and could prove basic competence. “From minute one, I was like, ‘All right, who can I transfer this ownership to? Who can I give this responsibility to?’” says Norbert de Langren, lead maintainer of Storybook, who initially planned to only serve in this capacity for a single year.

#### Provide radical transparency.

Maintainers often stated that they tried to push everything happening in their projects from private to public channels to achieve 100% transparency. Most intuitively understood that open source communities expect transparency. For the most part, private contacts came when private companies wished to submit code changes and wanted to understand the best way to bring this about. Maintainers discouraged side-channel discussions such as this. Radical transparency is especially important for two other reasons. First, it simplifies succession and continuity, and second, it enables the entire community to participate in decision-making.



*“Any open source projects where there’s a commercial company that’s supporting the project— it needs to be one community. It can’t be, you’ve got your commercial side, and they’re doing their own thing, and you’ve got your open-source community, and they’re doing their own thing. That doesn’t work because you’re defeating the purpose of things being open.”*

—GARETH GREENAWAY, SALT

#### Distribute power and decision-making.

Until 2022, the Jupyter Project used community-wide rough consensus for making key decisions. While consensus worked well in the initial stages of the project, as it grew in complexity and number of participants with over 1,500 contributors and over 100 GitHub repositories, building broad consensus became unmanageable. Many parts of Jupyter required specialized knowledge to understand and make decisions. According to Jupyter co-creator and maintainer Granger, even sub-projects with 50 or 100 contributors and maintainers were struggling with consensus. This was slowing innovation.

As a result, the project leaders struggled to make decisions when consensus was inadequate. This led to community member frustrations and complaints, which in turn contributed to the burnout of project leaders. After public discussions and deliberation, Jupyter adopted a modular governance structure with an Executive Committee, a Software Steering Council, and Working Groups to parcel out decision-making. “Our new model really works hard to make sure that we have a good balance of power checks in place, clear decision making processes, accountability mechanisms, and that we handle conflicts of interest in a responsible and transparent manner. All of this is to make sure that companies, individuals, and nonprofit organizations can come together and work on Jupyter in a neutral, collaborative context,” explains Granger.

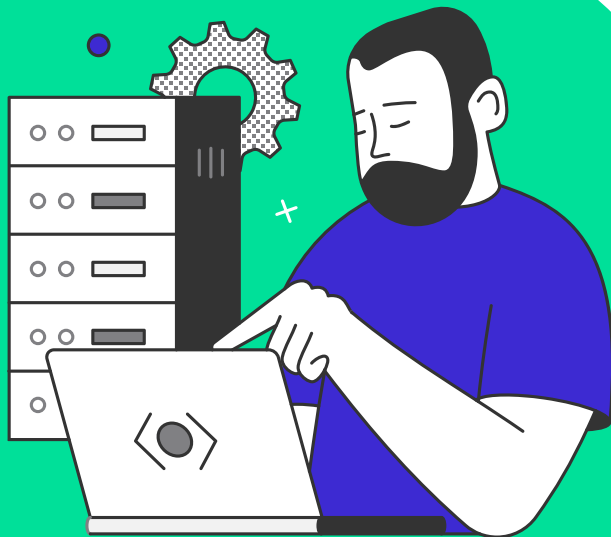
#### Maintain clear neutrality in community management.

This is particularly critical for projects controlled by a single organization or company. Experienced maintainers spoke of striving to maintain a neutral view of all contributors, even to the point of ignoring the identity of submitters of issues and PRs. Some maintainers acknowledged that if a contribution comes from a regular known for working on a particularly complex area, they may flag it. But, broadly speaking, they strove to treat all contributors and all

PRs filed with fresh eyes and on an equal basis. A critical piece of maintaining neutrality is demonstrating perception and awareness of what neutrality should look like and how a neutral party should behave. This entails both putting in the core work of maintaining a project in a very public manner as well as encouraging project development that is clearly non-zero-sum, even to the point of making a project more commercially useful beyond the controlling company.

Explains Phil Estes of Amazon, a core maintainer of the containerd runtime engine, “By doing generally helpful community chores, often referred to as ‘chopping wood and carrying water,’ many times you end up with credibility and trust that garners leadership in the project, such that when it’s time to say, ‘Hey, project maintainers, we think this would be a cool way to make the project more observable or measurable, or this feature would be really valuable for these specific use cases,’ those discussions get a lot easier when you’ve been a critical piece of just keeping the project healthy.”

That said, for many key critical open source projects, there is often difficulty finding independent or hobbyist community contributors with the time afforded those working for major vendors. This is a reality for projects that operate at the lowest layers of infrastructure, such as toolchains and runtime environments, which require more niche skills and could be viewed as less glamorous



than larger projects such as Kubernetes. “It’s rare to get someone who just shows up and says, ‘Oh, I’m here, and I want to work on containerd. I don’t work for a vendor who has container offerings, but I just love this technology,’” explains Estes.

### Use test sets to enforce neutrality.

Another piece of advice from maintainers is to use test sets as forcing functions for neutrality. “Meta understands the importance of PyTorch as a project, but another thing that protects us from conflicted interests between internal customers and external customers is a good test set. Unit tests that we run in open source, that we also run internally, provide a very good proxy of what the community needs versus what the company needs,” explains Nikita Shulga, a PyTorch core maintainer.

## Documentation

Creating a culture of strong documentation is essential at every stage of open source project development. Documentation enables smoother onboarding for new contributors and project users alike. As a project scales, documentation becomes a key tool for communicating not only the “hows” of project code but also the architectural philosophy behind the engineering approach. While users and contributors may not read the documentation before contacting project maintainers and contributors on Slack or in GitHub, the documentation can simplify support tasks by enabling cut-and-paste responses to common questions. Incomplete documentation, particularly for commonly requested information, not only erects barriers to adoption but also generates unnecessary work for already overworked project maintainers and contributors.

The significant majority of project leaders interviewed for this study expressed concern that their project needed to improve documentation. Recruiting and retaining good documentation contributors is a challenge. A second problem frequently cited is the wish to overhaul documentation competing against merging

new code. Additionally, they cited problems or lack of capability of documentation infrastructure, explains one of the core contributors and maintainers of the Apache Maven project:

*A full reorganization of the whole documentation is exactly the type of contribution nobody is able to get because there are people who perhaps could have the knowledge to change everything, but we would need infrastructure. We would need to rework the contents. In fact, nobody has the full ability to reorganize absolutely everything. It is as it is. I worked a lot to improve it, and at the moment, yes, it keeps working as it is.*

Projects that have built more advanced and systematic approaches to documentation undertake a variety of steps to improve the process and attract more contributors. Here is a summary of tactics used by successful maintainers to address documentation challenges.

**Make documentation a first-class citizen in the project hierarchy.**

This usually means a set of very visible policies and practices to demonstrate to the community that documentation is critical. One approach is for project creators to sit on the documentation committee and be involved in documentation efforts. Another is to include documentation goals and metrics as part of overall reporting efforts for a project. According to Rachel Lee Nabors, former documentation manager for the React Project at Meta, project founder and maintainer attitudes toward the docs team is a good indicator of the longer-term health of the project. She feels that when project founders and core maintainers take an active interest in documentation, that increases the likelihood that a project will continue to grow and attract new users.

**Hire a documentation coordinator.**

Because it is an activity so largely unloved in the engineering world, documentation that relies solely on volunteers tends to

struggle and suffer from a lack of resources or high turnover. Most of the critical projects have a full-time, paid coordinator or documentation leader or a paid project maintainer who is partly or solely tasked with documentation.

**Either recruit engineers for documentation tasks or create an educational process to nurture potential doc writers.**

The reality for complex projects is that documentation writers must have a strong technical understanding of how the code works. According to Nabors, many of the best documentation writers have engineering backgrounds. Some engineers actually enjoy documentation because they enjoy the creative act of writing.

**Require that contributors properly document their PRs as part of the merge and build process.**

Salt uses a flag in its development process to ensure that all code submissions and patches have fulfilled documentation requirements. “The way Salt is structured, every module, every function within a module has a docstring. If we see someone contribute a module that has functions that you would run via Salt that don’t have a docstring, that pull request is not going to get merged,” says Gareth Greenaway, Salt maintainer. The Linux kernel follows a similar policy. That said, documentation of code is only a portion of the required documentation, but it provides a necessary foundation.

**Formalize documentation sprints, forums, and other efforts.**

Many organizations employ “doc-sprints,” where engineers take a break from coding to help write docs. But few make doc sprints a regular event running on a calendar and even included in overall project planning.

**Make sure documentation writers receive public praise for their efforts from lead maintainers.**

Mentioning all the documentation contributors in release notes, shouting out to them in project meetings and online events, and giving documentation teams their own swag are just a few of the ways that projects can reward “docstars.” Calling them out on social media is another low-cost, high-impact way to show appreciation for documentation writers.

## Funding and other forms of support

How to support OSS creation is a controversial topic. Most open source projects do not directly collect revenues for services or products; many that do are controlled by a single company. It is important to separate support into two buckets: financial support of the contributors and maintainers themselves in the form of salary or sponsorship or business revenues and support of the project and its organizational requirements, including operations, marketing, infrastructure, and finances.

Financial support of contributors and maintainers was a concern and problem for only one of the interviewees of this project. Among critical open source projects, the majority of maintainers and core contributors enjoy full-time employment. In most cases, the employment allows them to spend a portion or all of their time working on their open source project. All but two of the interviewees for this project enjoyed sufficient financial support for their open source activities, primarily in the form of full-time employment. Two of the interviewees had been or were currently maintainers of projects that relied primarily on sponsor dollars to pay maintainers and to cover operating costs. One of the maintainers helped found a consultancy that offered consulting and services focused on the open source project the maintainer had worked on. Most of the maintainers worked at larger organizations, where part of their explicit duties was to continue contributing to or working on the project.

*“There’s a class of folks who want to contribute to documentation, but they just don’t know how because documentation change also means a GitHub commit in the Ceph project. There’s that class that wants to complain about the documentation but can’t get to the right audience. I created DocuBetter to bridge that gap and help both of those types of people be heard and helped.”*

—NEHA OJHA, Ceph



In one instance, a maintainer of critical open source projects that is a solo project, expressed frustration that one of their projects required a significant upgrade but that current mechanisms for funding open source made direct funding to him a challenge. Equally important, the maintainer noted that he did not even know how to properly price the upgrade work because he would not understand the full scope without additional research.

Broadly, LF Research found that successful projects behaved in a fashion similar to a business or a nonprofit organization. This meant software development was the first priority but projects also put considerable ongoing efforts toward fulfilling other operational requirements, including fundraising, marketing, operational support, and infrastructure. Numerous organizations and startup businesses are seeking to address one or more of these areas; the majority of these efforts focus on increasing sponsorships and



building funding sources. Fewer organizations focus on other operational aspects, which are inherently less interesting to sponsors and other funders. Several respondents noted that, globally, governments are only starting to recognize the importance of supporting open source projects with grants to foster the development of both technology tools and tooling for ongoing R&D in biology, materials science and chemistry, applied physics, and other relevant disciplines. Interviewees identified established the following best practices for funding and support.

**Ensure there are regular sources of project funding or support.**

At the risk of stating the obvious, the financial viability of a project is key to a successful project. This can take various forms, including inside funding at a company where a project is solving a problem but is not a core part of the company product; indirect funding through academic grants or ongoing research by academics (multiple key open source projects launched as academic research projects, including Ceph, Apache Spark, and others); direct funding through consortia or foundations; and direct funding at a company that uses the project as part of its core product but views a community and open source as additive and complementary to its business model rather than purely competitive (Chromatic is an example of this). For solo and unaffiliated projects, potential sources of funding include sponsorships (GitHub sponsors, LFX, Open Collective), paid support and dependency models (Tidelift), and one-time foundation or industry

grants (OpenSSF grants to OpenSSL and other key open source security practices).

**For small independent projects, determine if they require funding and, if so, designate a funding lead or set up a supporting business.**

Of the three small independent project maintainers interviewed, two had a person working for the project as an employee or in project leadership responsible for funding. The third did not prioritize work on their own open source projects over their day job, which did not include working on those projects. The two smaller projects that did have a full-time fundraising maintainer (who also handled other non-code tasks) were able to pay primary maintainers a living wage based on sponsorship and other forms of support. The lead cURL maintainer dedicated part of their work hours to helping paying clients solve cURL issues or designing and coding features clients wished to have included in the cURL code base. While not a massively scalable business model, it did generate sufficient revenue to maintain the lead maintainer's paycheck and to earn a profit for the maintainer's employer.

**Small independent projects should offload as much administrative work as possible.**

The reality of any open source project is that it is a nonprofit organization and must check all the boxes required by these organizations, including handling finance, marketing, infrastructure, internal IT, and more. Developers working on open source projects are not COOs and rarely enjoy that part of the job or responsibly managing an open source project. This can be accomplished through joining foundations, signing up with organizations that focus on providing operational support to open source projects, and utilizing free automation tools in platforms such as GitHub to perform key operational tasks, ranging from the deployment of new code builds to license selection.



## Diversity

As with other sectors of tech, diversity remains a challenge for the significant majority of the interviewees. Few of the most critical open source projects have a significant representation of diverse maintainers and contributors. Some have virtually none. While all maintainers interviewed stated they wanted to improve the diversity of their maintainer and contributor base, many projects did not have defined diversity strategies, and some pursued limited or no programs to increase diversity. That said, some diversity efforts are now commonplace, such as using inclusive language (see Contributor Experience).

Projects of smaller size and operating in more complex areas of programming (runtime environments, databases) do not require large contributor bases. In these smaller projects or teams, it is often hard to sustain formal efforts to improve diversity; maintainers tend to have to juggle multiple priorities. The most pressing need is shipping code because there is no bench of potential contributors eager to enter and contribute to the project. This problem is particularly acute for projects written in older, more technical languages such as C or Java. "It's an ancient language for most people. It's a very niche language over there, and it's mostly just old people working on it," explains Stenberg, who says cURL does not have a defined diversity strategy beyond periodic outreach. "We would love to do better with getting new contributors and creating more diversity. I have posted on Twitter asking for help. We have asked the community. I am very open to any suggestions."

Programs to improve diversity, such as Google Summer of Code and Outreachy, represent additional responsibility and time commitments for already time-strapped smaller projects, according to maintainers of these projects. Larger projects and projects affiliated with foundations or umbrella groups or larger corporations are more likely to pursue these outreach programs. Maintainers report mixed results on the quality of interns and contributions.

More advanced and developed projects had specific diversity

strategies, which included some best practices for building more diverse project leadership. Some of those best practices included:

**Make diversity and inclusion a first-order goal of a project.**

Many critical projects treat diversity and inclusion as a bit of an afterthought. They lack a standing committee or defined efforts to build diversity. They may have goals or aspirations but few specific mechanisms in place to advance diversity. Forward-thinking project maintainers seek to include diversity and inclusion as a key goal of the project. For example, Jupyter had significantly elevated diversity and inclusion efforts to the highest levels of the project in its recent governance changes. "Our new governance model has a DEI standing committee that is a key part of the governance model," says Granger, one of the creators of the project and a lead maintainer. "That body has a seat, for example, on the Software Steering Council that makes decisions about the overall direction of Jupyter's software. We are prioritizing diversity and inclusion, and we haven't always prioritized it at this level."

**Pair mentoring with diversity efforts.**

Recruiting contributors who can later become core maintainers is only half the battle. Most lack experience in open source and can benefit from active mentoring by maintainers, who can explain the conventions of the paradigm and help them navigate the processes and group dynamics. "There are a few challenges in mentoring this large number, and they require careful planning to minimize the overhead," explains Khan of the Linux Project, who mentors 13 individuals. The office hours are for answering questions, sharing resources, and demonstrating tools and debugging techniques as required. For Khan, some of these new contributors have turned into mentees working on becoming more regular contributors as part of her work on the LFX Mentorship program at the Linux Foundation.

**Participate in third-party programs to boost diversity, such as Outreachy.**

There are a handful of programs dedicated to fostering diversity in technology companies and roles. The most prominent is Outreachy. The Linux Foundation also funds a variety of diversity scholarships. “I believe in Outreachy. I think it’s a fantastic program for being able to bring in a wide variety of contributors and other things,” says Abbott. “I’m thrilled to see that Fedora continues to host interns with Outreachy pretty much every round they can, and they work in a wide variety of areas, including not only engineering but also design and documentation.”

## Preventing maintainer burnout

The job of maintaining a vital open source project can be exhausting and draining. Each interviewee was familiar with the idea of maintainer burnout. Several maintainers mentioned that part of burnout comes from the different nature of open source development. All activity is public and subject to comment and scrutiny. Projects that rely solely on consensus for decision-making among participants and leadership often require additional work and interactions to build consensus and may accelerate burnout. “I think that there’s a certain magic to a project like Node.js, for example, that does the open model, that has a truly openly governed steering committee, with an open governance system,” says Myles Borins, a maintainer of the npm package repository and Node.js. “But I think to a certain extent, the ambiguity that exists and the difficulty of really balancing a consensus model with a lot of people who tend to be conflict diverse or just want to be nice all the time results in patterns that are likely to burn folks out.”

Most interviewees had created specific “survival” practices and strategies. These are not complicated or detailed; most are common sense. That said, it remains useful to survey these practices. Some burnout prevention methods included:

*“There’s like a Maslow’s hierarchy thing going on there. If a maintainer has enough money, they don’t have to overwork themselves at a different job, and they don’t have to worry about medical problems and housing and food and things like that, then I think that the risks of burnout drop considerably.”*

—JORDAN HARBAND, MAINTAINER  
OF QS AND es5-shim

“

**Recognize you will never “finish the job.”**

Until a project is sunsetted, open source work is never finished. There will always be PRs and issues waiting to be read and responded to. Successful maintainers recognize and accept that their work is never-ending. “Always just be realistic about what you personally can’t accomplish. It’s okay to balance that with other things. It’s okay if you don’t complete everything on your to-do list because open source maintainership can fill an infinite amount of time if you let it,” explains Abbott.

**Accept that open source is always on, and embrace the hybrid lifestyle.**

Some maintainers that did not experience burnout designed their lives to work around the flow of open source. They never really turned off from their projects, but they did take advantage of the flexibility of open source to maintain healthy external pursuits and spend time with family and friends. Jordan Harband, maintainer of ‘qs,’ for example, elected never to take a vacation

from his open source maintainership and governance work, but he built a lifestyle that enabled him to spend time with his partner and children, exercise, and hang out with friends. This is not for everyone, he notes, but it worked well for him.

**Working on open source projects that are rapidly expanding and require administrative overhead as an unpaid hobby is not advisable.**

This was the general advice from interviewees, the majority of whom were well paid to work on fast-growing open source projects. Many worked on open source projects on the side for fun, but none of those projects required the same levels of administration and coordination work as their paid open source work. “If this is something you’re not necessarily getting paid to do and you are having to spend extra hours outside of your working hours doing stuff that’s not fun, it might be a different story in terms of causing maintainer burnout,” notes Eli Uriegas, a PyTorch maintainer focused on build and release tooling.

**Constantly look for efficiency hacks to optimize and automate your workflows.**

Super maintainers tended to have bespoke setups to filter project activities and allow them to focus on the activities and issues that mattered the most. This may include an aggressive use of email labels, filters in GitHub, the use of bots to automate workflows, or other mechanisms to automate processes or enable greater focus. For some maintainers, such as Tobias Koppers of Webpack, the best way to do this is only to respond to communication in only one channel—in his case, inside of GitHub.

**Set boundaries and stick to them.**

Seasoned maintainers said that they strictly stick to boundaries on various parameters. For example, most refuse to engage in Twitter conversations about

the project. Most avoid private email conversations about their project, wishing to route all discussion into publicly visible GitHub repos or email listservs. Some maintainers simply ignore comments or messages outside of the project’s GitHub repos. Additionally, many maintainers set workday boundaries to ensure they spend time with friends or family. They found that once the community understood the boundaries, they respected them and tended to follow them.

**Step away if you feel burned out.**

Multiple maintainers said that when they begin to feel burnout, they take a vacation or they take a break from working on their project. In part, this came with cultivated self-awareness of what burnout feels like. Most carve out exceptions for critical periods, such as big version pushes, annual project planning sessions or online events, and vulnerability remediation efforts.



# Conclusion

Over the course of interviewing 30+ maintainers, LF Research heard many amazing stories and learned how some of the most successful projects effectively navigated the myriad challenges of founding, nurturing, and growing an open source project. LF Research hopes that these insights enable future generations of maintainers by documenting patterns and anti-patterns of successful maintainership and project formation and management. These patterns can provide detailed situational guidance for many of the common problems faced by open source projects and their maintainers, as well as ideas on how to surmount or, even better, preclude these problems from developing and becoming significant concerns. In the detailed anecdotes and suggestions, LF Research aims to provide maintainers with a basic tool kit for managing their projects as well as ideas for creating a positive work / life balance. This report only scratches the surface of an immense topic; in the interest of brevity, many instructive maintainer stories from our research did not make it into this paper. That said, from these interviews, LF Research garnered several specific action items for maintainers that can improve both their project experience and project code quality.

## Determine your project attributes.

Identifying your project attributes from the four categories listed below is key for creating a maintainer strategy (see **TABLE 1**). For example, a small but complex project with high criticality (OpenSSL is a good example) will likely need to think about both recruiting highly skilled maintainers and securing funding sources from corporations or others that rely on the project. And any project with high criticality benefits from a transparent, community-based governance process that is designed and implemented with community input—as early in the project life cycle as possible. Very large, complex projects that are well-funded, like Kubernetes, face more of an issue of promoting governance and neutrality. Medium criticality and moderate complexity projects may struggle

to attract maintainers, so they may do well to seek out a corporate sponsor. Each combination of attributes comes with its own recommendations, and there is no one-size-fits-all playbook. That said, common patterns for specific attribute types can help inform project strategy and direction. Projects will change and evolve over time. But even at inception, it is possible to have some idea of where a project sits with regard to the four attributes.

**TABLE 1**  
**PROJECT ATTRIBUTES FOR CREATING A RELEVANT MAINTAINER STRATEGY**

Size	Small	Medium	Large
Supported	No Funding	Some Funding	Well Funded
Complexity	Simple	Moderate	Complex
Criticality	Low	Medium	High
Lifecycle Stage	Startup	Fast Growth	Mature

## Create a strategy and road map based on your attributes.

Recognizing that often open source project management is ad hoc, and too much structure can become a blocker to productivity, maintainers will find it useful to create a rough strategy for the project to help focus activities and guide contributors and co-maintainers. This strategy document is worth revisiting on an annual basis to ensure that activities match attributes. For example, if a project moves toward becoming a critical

dependency in the technology ecosystem, then a project may need to prioritize additional security activities. If a project is clearly moving from early stage to fast growth, then the maintainers may need to spend more time building onboarding infrastructure and ramping documentation. Again, there is no perfect recipe, but a mindful approach to maintainership regarding attributes and stage can enable focus on what matters in what is always a noisy process.

## Identify key metrics for your project, and track them regularly.

Project and community health are highly subjective and dependent on the maintainer and community definition of success. For example, a front-end framework primarily backed by a large company may care less about the pipeline of new contributors versus whether contributors are constantly submitting bug reports and suggesting fixes.

And projects that are deep in the stack and complex to code for may prefer to measure existing community performance—ship dates, code quality, etc.—rather than community growth. That said, determining the health indicators for a project and measuring them is a key step for better project management and more productive maintenance. [The CHAOSS Project](#) offers a laundry list of community health

metrics that provides a palette to choose from and a great starting point for maintainers.

## Identify best practices for your project.

Open source project formation and maintenance can be overwhelming. Focus and clarity are critical. The preceding three pieces of guidance lay the framework and strategy. Putting in place the tactical steps and processes to impact project health and drive project improvements in the desired areas works best when project leadership and the community cooperate to lay out playbooks, best practices, and specific ways of working. It is possible for projects to lay out detailed guidance across a wide variety of areas, but for all but the largest and best-resourced projects, maintainers may want to focus best practices on the areas and metrics identified as being most impactful. For example, Salt is now at a scale where it wants higher quality code and less code overall in the core project, with more functionality moving to Salt-community-managed extensions. To match this goal, Salt has made test compliance a mandatory part of the submission process, versus optional before when the project was more focused on pulling in new contributors to the core project's code base. This is just one small example that highlights how you can leverage best practices to impact-focused areas of a project.

LF Research hopes that this will create a starting point for conversations and that, with the help of the growing maintainer community, LF Research can grow this body of knowledge, heuristics, and observations to cover a wider range of situations and tasks than this initial effort. The open source maintainer plays a critical role in accelerating global innovation and building enterprise, government, and nonprofit technologies that will solve global challenges and improve people's lives. Creating a body of reference knowledge and a larger pool of competency in this discipline will ultimately make open source more useful and successful by helping maintainers and contributors help themselves, building on the shoulders of what we have learned from those who came before.



# Acknowledgments

This report would not have been possible without the help from numerous open source project maintainers who took the time to talk with us about their work and share their wisdom and insights. We wish to thank:

- Rachel Lee Nabors, Brian Granger, Phil Estes (Amazon Web Services)
- Myles Borins, Andres Freund (Github, Microsoft)
- Eli Uriegas, Nikita Shulga (Meta)
- Shuah Khan (Linux Foundation)
- David Edelsohn, Neha Ojha (IBM)
- Siddesh Poyarekar, Carlos O'Donnell (Red Hat)
- Anil Sharma, Gareth Greenaway, Pedro Algarvio, Megan Wilhite (VMware)
- Norbert de Langen (Chromatic)
- Daniel Stenberg (wolfSSL)
- Laura Abbott (Oxide Computer)
- Tobias Koppers (Vercel)
- Joel Orlina, Jason Swank, Hervé Boutemy (Sonatype)
- Sterling Greene (Gradle)
- Liz Rice (Isovalent)
- Ralf Gommers (Quansight)
- Henry Zhu
- Jordan Harbend

## About the Author

Alex Salkever is the co-author of four books on the impact of technology on business and society including, “The Driver in the Driverless Car” and “Your Happiness Was Hacked.” He worked in senior leadership roles in product and marketing at numerous technology companies and also served as the technology editor for *BusinessWeek*. A longtime open source advocate, he has worked with the Linux Foundation on special projects since 2016.



Founded in 2021, [Linux Foundation Research](#) explores the growing scale of open source collaboration, providing insight into emerging technology trends, best practices, and the global impact of open source projects. Through leveraging project databases and networks, and a commitment to best practices in quantitative and qualitative methodologies, Linux Foundation Research is creating the go-to library for open source insights for the benefit of organizations the world over.



Copyright © 2023 [The Linux Foundation](#)

This report is licensed under the [Creative Commons](#)



[Attribution-NonCommercial 4.0 International Public License](#).

To reference this work, please cite as follows: Alex Salkever, "Open Source Maintainers: Exploring the people, practices, and constraints facing the world's most critical open source software projects," foreword by Shuah Khan, The Linux Foundation, July 2023.