THE OPEN SOURCE WAY 1.99

Contributors

Version 1.99, 2020-11-17: This preview release contains opinions

Table of Contents

PREVIEW RELEASE

This preview is available in HTML single page and PDF.

Bugs (mistakes, comments, etc.) with this release may be filed as an issue in our repo on GitHub. You are also welcome to bring it as a discussion to our forum/mailing list.

Chapter 1. Presenting The Open Source Way

An English idiom says, "There is a method to my madness.".[1] Most of the time, the things we do make absolutely no sense to outside observers. Out of context, they look like sheer madness. But for those inside that messiness—inside that whirlwind of activity—there's a certain regularity, a certain predictability, and a certain motive. A method.

A method is a *way of doing something*, a particular manner or style of practice. The practice of open source community management can seem like its own form of incomprehensible madness, especially to someone who is unfamiliar with how open source software gets made, or those who find themselves staring down the prospect of having to coordinate others to join in making it. What this guide hopes to provide is some insight into an ever-evolving methodology, a form of practice. At its core is a simple question: What are your preferences for practicing the open source way?

But beyond that, this work is really digging into the question of *why* someone might do the work this way. Why use this or that method to organize the madness?

So from this introduction you should take two important points:

1. There is a *what*, a *how*, and a *why* to the practice of successfully and sustainably creating and maintaining open source software.
2. This guide distills opinions from a sizeable group of open source practitioners, the kind of people always having to answer the question of, "Why do we do this open source thing the way we do it?"

In particular, this guide is a living document that endeavors to gather and spread a diverse group of voices who are similarly (and somewhat-opinionatedly) collecting the best ways to create and manage an open source software community.

These voices are all part of this community of practice around open source software community management.

1.1. The shape of things (i.e., assumptions we are making)

When you look at this guide's table of contents, notice the organizational flow: from user to participant to contributor. This arrangement specifically tracks our view of how a community comes about and grows. And it all starts with a need. If you've worked in open source projects and communities, you've likely seen the cycle many times. It goes something like this:

1. A person or group addressing a problem creates the first iteration of software that offers one solution to that problem.
2. Other people with similar needs start using it, forming the core of a userbase.
3. From the userbase arise enthusiasts, people who promote the software and the

community emerging around it (that is, the values and principles that project seems to espouse).

4. Eventually arising from the users and the enthusiasts come people who are interested in (and have grown capable of) contributing to the software in some way; they ensure the project remains active and responds to the changing nature of the problem for which it was originally developed.

Thus, the primary and continual focus for a project is to *take good care of your users*, for some of them will become your contributors, and you want them to do it out of love more than resentment.

A key element of all this is recognizing the value in contributions of all types, at all levels. Rather than valorizing only a single type of contribution, the best practice is to treat all manner of gifts to the project as equal. These contributions comprise content and code, hours and effort, and all types of broad contributions for the project via events, discussion forums, infrastructure maintenance, and so on.

When you lower barriers to participation, you make your project welcoming at all levels. This doesn't mean you *remove all barriers*—just lower most of them appropriately. Appropriate barriers are necessary for someone receiving keys to systems, for example, and these barriers are different from those allowing only certain participants to contribute a help article or a patch to the configuration management system.

1.1.1. Our core opinion

There is an opinion at the core of this work, that explains how the content is laid out, how the steps on the path are revealed so that you can find yourself wherever you are in your journey of open source practice.

> Focus on your end users and lower the barriers to participation all around.
>
> Contributors arise from participants, who all started as users.
>
> Make your software wildly successful by having a user-centric initiative, while making the community open and welcoming those users curious enough to investigate how the software gets created.

The pathway of this guidebook therefore is:

1. **Attract users** to your software because it solves the problems they have; then
2. **Guide participants** who care about open source and sharing good solutions so they can be effective enthusiasts for your software; and
3. **Grow contributors** from this rich, fertile user and enthusiast base, by making sure when any of those folks look into how the software is made, they see themselves represented and can imagine how they could fit in, too. Make sure when they get to the contributor party, the barriers are clearly lowered and welcoming.

Think of it this way: It's one thing to let people know about your great dance party. It's another thing to empower those dancers to advocate for your way of throwing a dance party. It's a very great thing to make the dance party able to evolve to welcome all manner of music and dance from the world around us.

To get the dancers who see your poster and show up is the first step to stick around, and it helps if they can see right away that this is a place for them and theirs, too.

You can see this pathway to project success doesn't necessarily emphasize expertise (or even existing skills relevant to the project) more than curiosity and willingness to try the software, to care about the software, and to help create the software. This does not mean that people don't bring skills to the project; they do, many of them right from first point of contact with the project. But it is easier to teach skills than to teach someone to truly care about your project.

In this journey, there are many best practices we recommend, including how to put together a plan for measuring your community's progress. We also list some specific areas to be careful of and watch out for, including a robust discussion on self-care and mental health considerations for community managers themselves.

Our best way of more-deeply conveying all this information is with stories. Aside from stories that naturally arise in the body of the chapter, there are other areas in the guidebook and in The Open Source Way community to gather all these stories of why.

1.2. Structure of this guide

Each section of this guide consists of one or more chapters.

A chapter represents discussion of a discrete topic or closely-related set of topics.

The idea is that you can open and read a chapter as a stand-alone experience without needing to read the rest of the guide to understand and act from the material in that chapter.

A chapter consists of several parts:

- An introduction
- Primary material, written in third-person point-of-view, focusing on a balance of what, how, and why content. The examples (stories) told here are brief and in the third person.
- Expanded stories come at the end of the chapter, and are one or more stories from the chapter author(s). These may be written in the first person—that may be the most effective way to make the point in many cases. These stories are modular, in that another story that makes a similar point but from a different situation can be used instead.
    - The idea is to start with a base story or set of stories in a chapter, and supplement or substitute other stories that are contributed to the project, such as through the interview effort.

- Each chapter contains a lexicon at the end that defines the special usage for terms within that chapter.

Finally, we're collecting stories from practitioners of the open source way, stories we know tend to illustrate more than one principle in action. The idea is to start with a base story or set of stories in a chapter, and supplement or substitute other stories that are contributed to the project, such as through the interview effort. Stories not interspersed into chapters will be collected at the end of the guide (for your endless uses).

1.3. A community of practice always rebuilding itself

What you are reading here is just one facet of the growing body of principles, implementations, and examples that this community is gathering, cultivating, and maintaining.

In the end, it's just one way to pull this material together (one method, you might say, of organizing the madness). We'll be updating this guide. We'll be issuing similar, new guides. And we'll experiment with other ways to understand and present this material.

But at the core—in addition to the *what* and the *how* that benefit your open source community—you will also learn to understand the *why*, and be able to spread those stories wherever you go.

[1] From "Hamlet" by William Shakespeare, Act 2 Scene 2: Polonius (aside) "Though this be madness, yet there is method in `t"

Chapter 2. New Project Checklist

This is a relatively simple checklist for you to consider when starting a new open source project, especially where the project may be starting small but wants room to grow. By *simple* we mean: this list doesn't propose a process or project management for accomplishing these items; in its most simple form it does not have definitions for items; it suggests what is to come without prescribing or demanding obedience to the list.

These are separated out into several different lists, depending on the area covered:

2.1. Goals of project

- [ ] Technical problem addressed by the project
- [ ] Intended users and value proposition
- [ ] Establish beginning roadmap w/milestones

2.2. Market positioning

- [ ] List of related/similar projects
- [ ] Why a net-new project?
- [ ] Key differentiators

2.3. Project name

- [ ] Compile & vet candidate list
- [ ] Logo design
- [ ] Legal review (if required)
- [ ] Reserve name (domain name, GitHub, social media handles, etc.)

2.4. Licensing & legal

- [ ] Document license criteria
- [ ] License selection
- [ ] Need ™ or other mark registration?

2.5. Governance

- [ ] Define officers & their responsibilities
- [ ] Org structure, voting reqs & process
- [ ] Rules for amending governance

- ▢ Rules for contribution, committer status
- ▢ Provisions for sub-projects & lifecycle mgmt
- ▢ Privacy policy
- ▢ Code of conductr
- ▢ Foundation membership options, if planned

2.6. Project infrastructure

- ▢ Mail tool (and moderators)
- ▢ Forum/Chat (and moderators)
- ▢ Doc repository (slides, planning docs, etc)
- ▢ Web conferencing platform
- ▢ Community calendar (tool + who will maintain)
- ▢ Public website and website maintenance
- ▢ CI/CD, dev & test environments
- ▢ Lab requirements, how they will be acquired
- ▢ Code contribution tools and process
- ▢ Project documentation platform

2.7. Ownership and financing

These items may be ignored if donating project assets to a foundation

- ▢ Website URL
- ▢ Logo
- ▢ Social media handles
- ▢ Web conferencing platform (if paid)
- ▢ Process for funding project needs

2.8. Launch planning

- ▢ D&I plan
- ▢ Community health and metrics vision/plan

Chapter 3. Communication Norms in Open Source Software Projects

3.1. Introduction

While the tools projects use to communicate continually evolve, best practices for communicating among project participants rarely change. This chapter will cover the basics of project communications and offer advice on serving a global audience by making the best use of particular tools and platforms. We'll see how certain norms and techniques for maintaining productive communication in your community will ring true everywhere, regardless of the communication tools or platforms a project chooses. And we will conclude with some observations on the ways ever-changing tooling and communication practices in open source projects may present challenges for the outside observer—particularly academic researchers—along with recommendations for mitigating those challenges in a way that benefits all project participants.

3.2. The evolution of communication platforms in free and open source software projects

As the number of free and open source software projects has increased over time, the communication tools projects use—both to connect with their users and to coordinate their developers and contributors—have evolved. Internet Relay Chat (IRC) was once the default choice for real-time project chat. Many large projects conducted development discussions solely via their electronic mailing lists, Freshmeat was the place to hear about new project releases or updates to existing projects. Discussions about feature design and project improvements occurred via email and were included in a project's mailing list archives.

Today, the landscape has shifted in multiple ways.

While real-time chat is perhaps less common today, projects have not forgone it entirely. Many projects have simply adopted services that are considered more user-friendly or accessible, for example, Gitter, Slack, RocketChat, and others. A project may even use a tool to bridge back to older chat systems such as IRC in order to help a community evolve to include newer services.

Some communities still employ email for asynchronous communications, and others have found they can achieve better results by offering their users the opportunity to collaborate in online forums, a format more familiar to many developers who began coding at the same time they were sharing news and perspectives on then-new and popular forum sites like Reddit and Imgur. Forum advocates find the visual layout more appealing, and forum administrators often prefer the medium's fine grained access controls easier to use for moderation. (For example, being able to stop all posts in a particular thread during a heated discussion is relatively easy via most forum software—not so with mailing lists.)

While Freshmeat remains active (now renamed to Freecode), free and open source software projects have also made extensive use of social media services like Twitter and Facebook to interact with their user and developer communities. Social media serves as an excellent vehicle for project marketing and keeping audiences abreast of major project developments.

Social media also provides a vehicle for:

- Advertising upcoming events
- Live streaming of content or talks
- Generating interest in a project
- Other forms of marketing and outreach that broadcast further than a project’s newsletter, website, or blog.
- And many users now expect to receive basic technical support via these platforms.

While mailing list archives may no longer serve as a definitive record of project discussions, GitHub's popularity as an online development and collaboration platform, has meant many projects now manage their development discussions via GitHub issues. GitHub’s issues can have *tags* to differentiate, for instance, between discussions addressing implementation of a new feature and those reporting that the software does not work as intended.

And yet while the tools free and open source projects use to achieve communication are rapidly evolving, the human needs for both information and a sense of connection have not changed.

3.3. Project communications: The basics

3.3.1. What great communication accomplishes

People must understand the "who," "what," "where," "when," "why," and "how" of working with an open source software community. Thoughtfully established and well-maintained communication channels enable open source project and its participants to:

- Establish shared understanding of what the software it is, why people may wish to use it, and how they can get started doing so.
- Educate the community about how to use the software and how to contribute to the project.
- Keep people informed about project events (like a conference or webinar) and developments (like a new software feature, the formation of a working group, or the arrival of new contributors).
- Preserve knowledge about the project’s decision-making norms and practices.
- Allow users to report an error with the open source project’s software itself and to submit fixes for errors they or others uncover.
- Build a sense of common interest and purpose amongst participants, and provide an

outlet for people in the project to socialize or "hang out" together online.

- Provide a location for contributors to the project to collaboratively work together, be it on documentation for the project or to discuss how a particular part of the code base should be refactored.

3.3.2. Know your audience(s)

As you develop project documentation and other informational resources, recognize this work has several different audiences. Understanding these audiences and prioritizing their unique needs can help you craft better materials.

For example, consider the basic distinction between project *users* and project *developers*. Someone interested in *using* the software will be interested in things like how to install it, how to configure it, and what they need to know to accomplish the task at hand (e.g., build a website). Developers will also be interested in such details, but will need additional detail if they are going to contribute effectively to the project.

If your project observes certain coding conventions, be sure to include those conventions in the developer documentation. Having a patch rejected by a maintainer because you haven't indented code a certain way can be rather demotivating. Users, however, likely won't need these details.

TIP	Develop a style guide for your project and make sure it is easy to find in your developer-oriented documentation. Not every project will create its own style guide, but it is best practice to note what coding conventions the project will observe. If the project uses a previously published style guide, make sure to link to it in the developer documentation. [2]

3.4. The virtual showcase: crafting your project's website

When people are looking for information about software that may help them solve a problem, their first stop will likely be the project website.

A project website needn't be intricately designed to be effective (though a more attractive site aids with project marketing). A thorough README.md file on GitHub can work well as a project website, provided the information it offers gives users and potential contributors a sufficient overview of the project. A basic project website should feature at least the following sections.

3.4.1. Project overview

Include a simple overview of the project in an easy-to-spot location on the project website. This project description should include what the code base is designed to do and what problems someone might solve by using it. Making this information succinct and easy to find is critical. Post a quick description on the project's home page so that interested parties immediately discover if the project is designed to meet their needs.

Think of this information as the opportunity to explain to someone who has never heard of your project why it could matter to them—in 30 seconds or less. For example, Drupal's about page describes the project this way:

> Drupal is content management software. It's used to make many of the websites and applications you use every day. Drupal has great standard features, like easy content authoring, reliable performance, and excellent security. But what sets it apart is its flexibility; modularity is one of its core principles. Its tools help you build the versatile, structured content that dynamic web experiences need.[3]

In this description—just a single paragraph—we learn:

- What Drupal is (a content management system).
- What a content management system is (a tool to build websites).
- Why Drupal is a compelling choice (easy to use, reliable, secure, and flexible).

Let's take another example from a popular project: Kubernetes.

When visiting the project home page, kubernetes.io, a visitor immediately sees the following explanation:

> Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.[4]

In this description, we immediately learn:

- What Kubernetes is. (It's a system for working with containerized applications, including deployment, scaling and management.)
- How Kubernetes is abbreviated. (Little details like this one immediately deepen the comfort level of new arrivals; no one is expected to already know an arcane acronym upon arrival.)
- Where Kubernetes was developed. (Google is noted as the originator of this code base, establishing the project as focused on enterprise applications and providing confidence that the software is well designed and maintained.)
- Kubernetes is open source. (A user can expect to use, run, modify, and share changes to the code base; any questions about barriers to entry due to licensing fees, procurement processes, etc., are dismissed.)
- The project values community engagement. (One can expect that contributions of code, documentation, etc., are welcome and encouraged.)

3.4.2. Getting started

The processes for creating good "getting started" documentation—occasionally called "onboarding documentation"—are outside the scope of this chapter. (Refer to the dedicated onboarding chapter in this guidebook for more detail.) Here, suffice it to say that open source project websites should feature a section aimed at helping new users and potential contributors get started using the software. Clearly labeling that section "getting started" or "new users" makes finding that section easy when people need it. Further differentiating between "new users" and "new contributors" in your onboarding documentation is even better, as these two audiences have very different needs. Clearly pointing to these resources for newcomers on the project website helps to keep the project's other communication channels—like the forums and real-time chat rooms—free from frequently repeated inquiries about how to get started.

TIP

In your project's "getting started" guide for new users and participants, include any information you can about other places those unfamiliar with the project can get help. For example, you may have a Slack channel called "newbies" staffed by folks who enjoy mentoring and helping people get started, whereas ongoing development discussions may take place in the "developer" channel.

3.4.3. Frequently asked questions

Another excellent location to feature basic information about your project is a frequently asked questions (FAQ) page. If project development is just beginning, a basic FAQ detailing what the project is, what the code base is used for, and how someone can get access to the code is sufficient. However, as more people join the project—new users, developers, documentarians, etc.—you will likely find yourself answering the same basic questions repeatedly. (And in the process you'll discover that many aspects of the projects are not as obvious to newcomers as they are to you.) These repeat questions represent opportunities to improve your documentation and to seek help from your community.

Keep your FAQ updated and easy to locate. But even better: ask community participants to help you improve it. When answering a question for a newcomer, be it via email on the project mailing list or in real-time chat, ask the newcomer to write up the question and answer for inclusion in the project FAQ. By asking for help from your community, you do several things:

1. Get help keeping your documentation relevant and timely.
2. Demonstrate that community contributions to the project are welcome and encouraged.
3. Invite further contribution from someone who has already shown interest in the project by asking for their help.

Ideally, newcomers would have the ability to edit the FAQ themselves. Sending instructions for how to edit the FAQ along with your request to contribute to it—thus lowering the barrier to entry—makes receiving a contribution more likely. If your project maintains a contributors list, make sure to include the people contributing to your FAQ in it. People love seeing their work and contributions (however small) acknowledged. Doing so gives contributors a sense of belonging and commitment to the project. People who feel their work is appreciated and respected are more likely to stick around and contribute to the project, whether by filing issues or adding new features.

3.4.4. Document project goals and non-goals

Your project's website should also make clear the *purpose* of the project and the *activities* the project has as its focus. People have difficulty understanding how they can best fit into a community and how they can contribute if they do not understand what activities are currently in progress and what is planned for the future.

One common tool to communicate these goals is a project roadmap. Even if your project does not yet have sufficient resources to develop this kind of roadmap, you should still find some way to ensure users and would-be contributors understand the project landscape. For instance, a weekly recap of project activities and planned activities for the coming week or month is an excellent start, and it's something you can offer through a quick blog or forum post. Such works are an excellent resource for newcomers orienting themselves to the project and are a wonderful place to point interested parties to learn more as part of their onboarding process.

Communicating your project's *non-goals* is equally important. Due to the vibrant nature of open source projects, it is only natural that someone will find a use for a project that the project's creators never intended and will wish to extend the project's capabilities to target that specific use case. If the project maintainers do not intend for the project to have a wider focus than what is already offered, letting these would-be contributors know this in advance will save everyone time and disappointment. In this era of easy forking, it is relatively easy for those who would use some parts of the project but not others to develop and maintain a code base that better matches their own needs—all without asking the maintainers of the original project to deviate from their intended vision and the project scope they've set.

Documenting non-goals is also particularly important for commercially focused projects, where a contributor's desire to create a feature as open source may be in conflict with vendor goals for creating proprietary features. Contributors may still choose to create that feature as open source, but they should know from the start that upstream maintainers do not intend to include their work as part of the project's code base. Some may choose to not implement the feature, knowing that a vendor is creating it for them; still others may choose not to implement the feature if they know it will not be included in the project's mainline source tree, as they do not wish to incur the burden of ongoing maintenance themselves. And others may choose to go ahead and create something that works well for them and release it as open source, regardless of whether the feature is incorporated into the project's main source repositories.

Most important here is that no one feels *surprised* by the direction a project will take. No project needs to accept every contribution, but having contributors invest time and energy into developing something only to discover it will not be accepted due to a conflict with an unknown roadmap (commercial or otherwise) creates tension in the community and a lack of trust in the project maintainers. It can even encourage adoption of open source alternatives to the vendor's product.

3.5. Not working as intended: Getting the most from the issue tracker

This section details how an issue tracker can be used as an essential communication tool.

3.5.1. What is an issue tracker?

An *issue tracker* (sometimes also known as a *bug tracker*, *issues list*, or *issue queue*) is a tool that allows people to submit reports when they encounter instances where they believe the software is not working as intended.[5] As a way to monitor pending tasks and allow for collaborative commenting and review of work in progress, some projects manage their entire development workflow via their issue trackers.

In this section, we'll discuss using an issue tracker for the purpose of reporting failures with the software. By reporting your issue using a project's issue tracker, you ensure maintainers who are looking out for problems see your report and act upon it.

3.5.2. Why file an issue?

While filing an issue may seem more cumbersome than simply asking for help in real-time chat, it is important to do for several reasons:

Project contributors cannot keep track of all conversations occurring across various platforms, but they can always refer to the issue tracker to improve the project.

Real-time chat and social media are ephemeral communication channels. The issue tracker is a purpose built tool for recording and reviewing problems with the software. Software projects often define their upcoming work plans by using their issue tracker as a key component—and perhaps their sole tool—to prioritize all possible areas to work on. (Simply put, the project's issue tracker is very often synonymous with the project's to-do list.) If your problem does not make its way into the issue tracker, it will likely not be addressed simply because a very busy person has forgotten the details of the problem. For this reason, you will often find that one of the first requests you receive when asking for help is to file an issue so the project maintainers can keep track of the problem.

Filing an issue allows you and the project contributors to communicate asynchronously about the problem, as all parties can refer back to and access the issue description and follow up comments at any time.

When you've uncovered a problem with the software, you might discover that the problem is actually the root cause of *another* problem, or there may be a way in which *several* problems are related. Issue tracking software allows project developers to group related issues together, which may aid in diagnosing a problem's root cause.

People often encounter the same issues with software, and many of them are filing issues with the project. Having multiple reports of the same problem can be very time consuming for the project maintainers, as they then need to respond to each individual reporter about work in progress. Fortunately, issue trackers make this process easier for maintainers by allowing them to quickly close issues by stating they are duplicates of an existing one (and then asking the bug reporter to track work-in-progress in the "original" report).

Project maintainers can then engage in broadcast-style communication to everyone experiencing the problem in one place, streamling their workflow while still helping everyone who needs assistance.

3.5.3. Make the issue tracker findable

Make sure the location of your project's issue tracker is prominently displayed in your FAQ, as well as in your usage and development documentation. If people cannot figure out where to submit an issue, they will ask someone in the project where to do so. Supporting well-meaning users by offering repeated answers to basic questions like this one can be quite time consuming.

Do yourself and your community a favor and make your issue tracker very easy to find!.

3.5.4. Use issue templates

Not everyone who uses your software will be familiar with your community's conventions for filing a useful bug report. To save you and the bug reporter time, offer an issue template to ensure you receive the information you need to reproduce the reported error and effectively triage it. For example, you may need to know what version of the software or what operating system was in use when an error occurred. If common information is required for reproducing errors, ask for it in an issue template.

Common fields in issue templates include a summary of the issue, steps to reproduce it, the actual behavior the user observes, the intended behavior for the software, and a request for log files or screenshots to help guide the issue reviewer in better understanding the bug report. Several issue trackers support templates for bug reports, including GitHub, GitLab, Redmine, and Trac.

If you find yourself asking for the same information over and over again in response to different bug reports, then congratulations. You have uncovered an area of your template in need of improvement.

3.5.5. Help wanted: labeling issues for clarity and encouraging contribution

Most modern issue trackers allow users to label issues they file, which can be useful for organizing project work. By differentiating between different types of requests—features for development, software errors, etc.—a project's maintainers can be more organized and triage issues more efficiently. Further, many people interested in contributing to open source software projects are looking for issues on which they can work to better understand the project's development mechanics. If you will actively use labels in your issue tracker, make sure to document the label definitions in your development documentation so those labels are used consistently (or restrict the addition of issue labels to project maintainers only). A list of labels that used inconsistently is no more helpful than a list of undifferentiated issues.

Labeling issues as "for newcomers" or "help wanted" allows project maintainers to flag issues particularly suited to contributors who have just joined the project. Labeling issues in this way shows that the project is prepared to onboard new contributors and that maintainers welcome community assistance in a particular area. Don't be afraid to file issues against project documentation, the website,or anything else you feel is amiss. If there's a place current and potential contributors can help make the project better, file these in your issue tracker with a clear label that shows them they can contribute.

Just make *very clear* (either in the text of the "help wanted" issue or via a link to other project documentation) how you wish others to engage with the project when working on these types of issues. (The Apache Subversion Issues page is an excellent example of clearly articulating needs to the user community before they file an issue.) It is best to encourage contributors working on these issues to engage with the project maintainers along the way, so their contributions have a higher chance of acceptance into the project. Nothing squelches a contributor's enthusiasm like showing up with a working solution to the stated problem only to be told their particular implementation will not meet the project's needs.

3.5.6. Communicate clearly and kindly

Whether you are a user of the project reporting an issue or a project maintainer reviewing a pull request, it is always important to communicate about the issue *clearly* and *kindly*. When a tool is not working, the person using it can become frustrated. Likewise, a person developing a project as a hobby is unlikely to respond well to demands on their time to fix a problem they do not have. Remember to be gracious and thankful in your discussions with other project participants, as everyone sharing their knowledge is contributing to the project's overall health and wellbeing.

3.5.7. When issues become the subject of heated debate

At times, the details of addressing a particular issue can cause tension or arguments within the community.

While healthy and respectful debate is part of any thriving project—software or otherwise—tempers can flare easily, and (as has been well documented)[6] people tend to behave with less civility online than they would in person.

If an issue has become especially contentious and discourse has become rude or inflammatory, restrict access to that issue for a stated period of time (say 24 to 48 hours) to allow people time to calm down, reflect, and state their argument in a more even-tempered and constructive manner.

3.5.8. Quick tips for filing issues

Thank the people creating the software for their time and energy, especially if you are new to the project. The individuals spending their (spare) time creating free and open source software for you to use are also people who want to know their time is valued and their work appreciated.

Include as much information as you possibly can about the error you have encountered. If the project uses issue templates, fill one out as completely as possible.

If you do not have the information requested or cannot determine how to get it yourself, simply note what you have attempted to do in order to get the information. These details help maintainers determine what they might need to do to assist you.

If a project does not use issue templates, look at other issues that have been "closed–fixed" or at merged pull requests to see how other people have filed bug reports. If the issue was fixed, chances are quite good that you'll be able to use these historical artifacts as examples of the sort of information necessary for reproducing an error. Replicate what you find in these reports and add more detail as you are able.

3.5.9. Quick tips for responding to issues

> "While the size and skill of the development community constrains the rate at which tickets can be resolved, the project should at least try to acknowledge each ticket the moment it appears. Even if the ticket lingers for a while, a response encourages the reporter to stay involved, because she feels that a human has registered what she has done (remember that filing a ticket usually involves more effort than, say, posting an email)."
>
> —Karl Fogel, Producing Open Source Software.[7]

Thank the submitter for filing the issue. Helping a project improve is an excellent contribution to that project's health. Further, by being gracious, kind and welcoming, you encourage continued participation and contribution from the issue reporter.

When closing an issue as "won't fix," explain why the issue will not be fixed. Maintainers shouldn't feel compelled to accept every pull request or fix every reported issue, but they should at least let bug reporters know *why* they won't be addressing certain issues.

In particular, if someone has submitted an issue along with code to fix a problem or implement a new feature, it is vital to tell them why their work has not been accepted by the project. Not doing so makes the contributor feel like they've wasted their time and should devote their energies to a different software project. In an ideal world, you are able to include a link to a published project roadmap that explains why the submission does not meet the needs of the project. (Refer to above.)

For new contributor submissions, fix minor issues with the patch yourself along with a note about what you fixed and why. Having a patch rejected for minor nits discourages additional contribution, and often it takes just as long to explain why a patch is being rejected as it does to make very small fixes. Such explanations are an excellent time to point contributors to additional project resources, such as your coding style guide, documentation on communication norms, etc.

For submissions coming in response to a "help wanted" issue, engage early and often with the person who has stated an interest in working on the issue. Doing so ensures that the contributor's submission will actually meet the project's needs. Further, by being available to and in regular dialogue with new contributors, you form a relationship with them that encourages mutual learning and increases the chances they will continue to contribute to the project's ongoing work.

3.5.10. Having development discussions and other conversations in the issue tracker

Conventional wisdom in the early days of open source software development held that communities should *not* carry on development related discussions in the project's issue tracker. Project communities instead preferred carrying on such conversations via mailing lists or in forums a number of reasons:

- People following the mailing list were able to comment and express their views and needs without needing to parse through the issue tracker
- Forum or mailing list conversations were seen as better for asynchronous and long-form communications, and popular issue trackers in the 1990s and early 2000s were an unwieldy way to engage in actual discourse.
- Discovering why a particular technical decision was made when those details were buried in an issue tracker was difficult, especially since the issue would be in a "closed" state once the decision was made. Looking for a closed issue to explain the technical direction of the project was considered counterintuitive.

With the rise in popularity of GitHub as a one-stop platform for online development work, conversations in the project issue tracker have become mainstream. GitHub's issue system visually mirrors the typically expected visual interface for forum software, making discussions in its system seem natural for those who began their development careers when online forums were first gaining popularity. Further, time and resources necessary for maintaining a Mailman instance or additional forum software as part of project infrastructure became cumbersome when all other infrastructure could be managed via a single tool. The addition of features such as the ability to "+1" an issue, set fine-grained controls on notifications for specific issues, and lock specific issues so that only project maintainers may edit it (while still allowing others to view the issue) made the move to discussions in the issue tracker more palatable and effective.

Nonetheless, interested parties should be able to follow discussions *outside* the project's issue tracker. Only the most deeply interested and invested individuals will rigorously follow every issue update, making engaging with the project difficult for casual contributors. While excellent search capabilities in online issue trackers make finding closed issues easier, the flow of an issue discussion does not fulfill the same function as a narrative description of a particular implementation or an explanation of why a certain decision was made. Note, too, that some maintainers who are most intimately familiar with the project—those who can recall specific issue numbers for particular discussions with ease—will not always be available to help with the project work.

Preserving the knowledge about decisions in an easy-to-access way:

- Saves time for people working to uncover the why of project processes.
- Saves time for maintainers so they need not rehash history regularly.
- Ensures that critical details on how and why decisions were made are always available even as project membership changes.

TIP	If your project carries on most of its development discussions in the issue tracker, consider taking some small steps to highlight these discussions in other ways that will be most accessible and discoverable to interested parties and wider audiences in general.

For example, you may summarize the discussion of the issue in a blog, forum post, or project newsletter, thereby preserving cultural lore for the project while simultaneously informing the broader community about the change. If the project does not maintain a blog or other publication mechanism suitable for such a communication, consider adding a list of watershed issues to your project documentation so newcomers can quickly become familiar with these critical topics, and for ease of reference for long-time project participants.

3.6. Communicating well across the globe

3.6.1. English as the lingua franca of the internet

Though we live in a world where more than 6,500 languages are spoken, for historical reasons the primary language used for communicating on the Internet—and therefore, in major open source projects—is English. For users and contributors who are not native English speakers, this fact can raise significant barriers to participation. There are a few steps projects can take to help those for whom English is not their first language to more effectively participate in the project.

3.6.2. Prominently recognize community resources available in multiple languages

Should your project be widely adopted and grow to the point that it hosts communication channels in more than one human language, make sure to list these resources prominently on your project's website. Include on the project's website a note that the project welcomes submissions from community members for resources that are not written in English. When the project receives such submissions act promptly to get them included in your project documentation.

As you would with any resource you point your community to, do your best to ensure the resource is helpful. If you are unable to vet the resources as helpful given currently available person-hours on the project, reference the fact that project maintainers have been unable to assess the resource themselves. Note you welcome feedback on its inclusion in the project's documentation.

3.6.3. Be kind and welcoming regardless of English proficiency

As this chapter has stressed numerous times, kind and gracious communication with all those who participate in your project should be a default mode of behavior for interactions. The same holds true when communicating with people for whom writing in English is difficult. If you have trouble understanding what someone is saying or asking for, ask clarifying questions to let them know you will be happy to help them. Don't simply ignore someone or tell them they are not welcome in the project due to limited proficiency in written English.

TIP	People who are not native speakers of English often begin their communication with the project with an apology for their poor English language skills. When receiving such a communication, thank the sender for writing and let them know you appreciate their efforts to communicate with the project. Where possible, point them to any resources that may be available to them in a language with which they are more familiar, for example a Spanish language forum or a Chinese language mailing list for the project.

3.6.4. Avoid idioms in written documentation

Every language features various phrases the actual words of which do not convey the intended meaning of the phrase, such as "over the moon" to mean extremely happy or excited or "raining cats and dogs" to refer to a serious downpour of rain. For those who grow up in a particular culture, the meanings of these phrases are obvious. But they can be confusing for those who lack the proper context for them. Rather than rely on idiomatic phrases, use plain language in written documentation to ensure the writing is most accessible to all people.

3.6.5. Expand acronyms and provide a glossary

While acronyms are a useful way for those completely familiar with a topic to save time and effort typing and speaking longer phrases, they obfuscate information for those less familiar with the topic. Further, acronyms are often overloaded, meaning that the acronym can expand several different ways depending upon the topic area. For example, someone completely new to a project may not understand that "LGTM" means "looks good to me" and that their work is therefore acceptable for merging into the project's source repository. If you regularly use particular acronyms as part of communicating in your project, take the time to create a quick glossary of these terms. Updating this glossary is a quick and easy way for volunteers to contribute.

3.6.6. Actively seek participation from localization volunteers

As mentioned earlier in this chapter, project maintainers should always be clear about *what kind of help* they're seeking from their communities. One key area in which to ask for help is the localization of documentation resources. Regardless of their skill level with software development practices, community members can actively grow the project and improve it by translating documentation, thereby making the project more accessible to more people and more potential contributors. Maintainers should be explicit about their desire to recruit contributors focused on localization.

3.7. Documenting your project's communication norms

When people approach a new project, they seek to understand how they can best engage with that project and interact with its community. Be sure your documentation clearly outlines your project's various communication channels.

Simply *listing* communication channels is not sufficient. Your documentation must make clear *what* each channel is used for, *when* to use a particular communication mechanism, and *how* people can expect to receive communications from the project and its community members through that channel. For example, a project with few maintainers who develop the work as a hobby project may wish to note on the project website that those developing the project do so in their spare time, so immediate responses to mailing list inquiries should not be expected. Someone whose hobby project is in use with enterprises might like to make explicit the notion that help is provided on a best effort basis. (Doing so sets expectations appropriately for those who are less familiar with how open source project communities function.)

3.7.1. Maintaining civil discourse

As has been discussed throughout this chapter, maintaining kind and gracious communications is vital for the project's ongoing health and well-being. While assuming everyone understands what "kind and gracious communication" looks like may seem natural, one cannot assume a consistent meaning for all participants, especially when dealing with a global audience. Project maintainers and community members do well to lead by example. But it sets an appropriate tone for the project to make an explicit statement about what constitutes civil discourse, what matters are off-topic for the project, and what is expected from anyone communicating with the project, especially about matters that may cause conflict .

From the Diversity Statement of the Dreamwidth Project:^[8]^

> "We welcome people of any gender identity or expression, race, ethnicity, size, nationality, sexual orientation, ability level, neurotype, religion, elder status, family structure, culture, subculture, political opinion, identity, and self-identification. We welcome activists, artists, bloggers, crafters, dilettantes, musicians, photographers, readers, writers, ordinary people, extraordinary people, and everyone in between. We welcome people who want to change the world, people who want to keep in touch with friends, people who want to make great art, and people who just need a break after work. We welcome fans, geeks, nerds, and pixel-stained technopeasant wretches. (We welcome Internet beginners who aren't sure what any of those terms refer to.) We welcome you no matter if the Internet was a household word by the time you started secondary school or whether you were already retired by the time the World Wide Web was invented.
>
> ….
>
> We have enough experience to know that we won't get any of this perfect on the first try. But we have enough hope, energy, and idealism to want to learn things we don't know now. We may not be able to satisfy everyone, but we can certainly work to avoid offending anyone. And we promise that if we get it wrong, we'll listen carefully and respectfully to you when you point it out to us, and we'll do our best to make good on our mistakes."

This excerpt from the Diversity Statement of the Dreamwidth project is an excellent example of how to document project communication norms. It is clear that everyone is welcome in the project, regardless of personal background, technical skill level, or focus for using the project. It makes it clear that mistakes will be made and that people are expected to use these imperfections as learning opportunities, not excuses to belittle other people. The statement tells users and would-be contributors that they may not always get what they want from the project maintainers, but that errors will be fixed and reasonable requests will be listened to, if not acted upon. Most notably, it's a list of expected *prosocial behaviors* rather than simply a list of what *not to do*. It identifies behavior the project maintainers and community members model and transforms it from the actions they take into words that help everyone understand what actions constitute good project citizenship.

3.7.2. Developing a project social contract

As projects document their community communication norms, they may find developing a project social contract to be a particularly effective exercise. A project social contract documents behaviors the project expects all participants to display and sets expectations for how project members will be accountable to others. The social contract is not necessarily a list of forbidden behaviors, but rather a statement about how members of the project will choose to self-govern for everyone's success. By undertaking the process of creating a social contract through dialogue, members establish empathy with one another and set the foundation for future conversations.

You can learn more about creating social contracts, including tips for doing so for remote teams, in The Open Practice Library.

3.7.3. Codes of conduct

Some projects use a code of conduct as a means to document their expectations around civil discourse. Open source projects that seek any outside contribution should always have a code of conduct. For projects that host events, whether virtual or in person, developing code of conduct specific language around events is also a best practice. Think of the code of conduct as an aspect of the project's social contract, one that includes the rules by which the community will govern itself and how each member will hold one another accountable for those moments when they could have behaved differently and achieved a better outcome. These rules must be understood and made explicit. Otherwise people will know neither what is expected of them nor whether the project is a place where they will feel welcome and comfortable contributing their time and expertise.

Refer to this guidebook's chapter on governance for more information about codes of conduct.

3.8. Successfully communicating with an open source project

So far, we've focused primarily on ways in which software project *maintainers* can ensure the best possible outcomes for communication in their projects. However, contributors, too, can take a number of steps to ensure they're communicating effectively with their favorite open source communities. Here are just a few:

1. Read the project website and documentation before jumping into discussions.
   a. Take the time to read about the project and understand its nuances.
   b. Demonstrate that you respect the time and attention of the people producing the project.
2. Do your research, and tell people you've done it.
   a. If you run into a problem using open source software, do what you can to solve the problem yourself. There's no shame in not being able to solve the problem, and it helps your bug report.
   b. Make sure to include what steps you have taken to resolve the issue when filing a bug report or asking for help in one of the project's communication channels. Doing so saves people time and energy, as they will not ask you to try something you have already done.
   c. Listing the ways you've already attempted to help yourself is a demonstration of

respect for the time and energy of the project's maintainers.

3. Practice basic netiquette.
    a. Most fundamental advice for communicating on the internet is applicable in open source projects and communities.
    b. For instance, avoid typing in all capital letters, as this style is read as shouting (and one would not go about asking for help by shouting at someone).
    c. Choose a username or screen name that is reasonable and approachable, otherwise you risk not being taken seriously by others.
    d. Wait a reasonable amount of time—say 24 to 48 hours—for a response to your inquiry before trying to get a response in a different communication channel.
    e. You may find Virgina Shea's oft-cited The Core Rules of Netiquette to be a useful resource if you are unfamiliar with the rules of engagement in internet communications.
4. Post questions and communications in the appropriate places.
    a. Encountering information in a place people don't expect can overwhelm them. For example, using a project issue tracker to let folks know you are hosting a hackfest next week is inappropriate.
    b. If the project has taken the time to let contributors know how and where to ask questions—and you should know this by following the guidance in the first item on this list—make sure to use the appropriate forum to do so.
    c. Demonstrating you have taken the time and energy to interact with the project's maintainers and other volunteers in the way they've asked shows you respect their efforts and, in turn, makes helping you be successful much easier for them.
5. Make the subject of your posts as meaningful as possible.
    a. When writing the subject line of an email or forum post, make your needs explicit.
    b. For example, a subject line that says "I think I found a bug" is likely to be acted upon more slowly than one that says "external display not recognized upon upgrading to version 2.2."
        i. The second subject tells the reader that they will likely find more detail on how to diagnose the problem, and that they are dealing with someone who understands the limited amount of time and attention the reader has.
        ii. The first subject does not differentiate the sender's problem in any way, and makes it difficult for your communication to be memorable to the reader.
        iii. The more useful the subject of your post, the more likely you are to receive a prompt reply.
6. Be kind and courteous in all your communications.
    a. Once more, let's stress that the key to effective communication in any project—open source software or otherwise—is thoughtful and gracious behavior.
    b. Do not show up at an open source project angrily demanding help for your

problems, send impolite follow up messages when you do not get an immediate answer, or otherwise be unkind to the people with whom you are communicating.

c. Do take the time to thank them for their help and for providing the project to you and everyone else.
    i. Remember you are communicating with other people, some of whom are spending their free time to write your free software.
    ii. Treat them with the respect and courtesy you want for yourself.

3.9. Evolving communications in open source projects and academia

While open source software now seems ubiquitous, we should recall that the free and open source software movements are still in their early stages. Development of the Linux operating system began in 1991. The Apache Software Foundation, steward of many of the world's most notable open source projects, was incorporated in 1999. Though 20 or 30 years seems like ancient history on the internet, it is worth noting that Ada Lovelace created the world's first algorithm back in the 1840s. Open source is still a blip (albeit a significant one) in a much longer technological timeline.

Due to open source's disruptive influence in the software industry, academic researchers have found open source software projects and their development methodologies particularly worthy of study.

However, as projects' communication tools and platforms have evolved, researchers' ability to access project data for the purpose of study has been, at times, diminished. For example, parsing IRC logs of a project's real-time chat often yielded fruitful information about a particular project, but as some projects have moved to other chat systems, such logs are no longer commonly available (nor has there been any guaranteed longevity of the project's chat archives, depending on which communication tool the project chooses).

When a project launches or consists of a small group of people working together, choices for how to communicate and where to do so often arise organically and with little consideration to the future impact of those choices. But project maintainers should thoughtfully consider how they can ensure the project's communications—which contain potentially rich sources of data and historical artifacts like lore and decisions histories—are effectively captured for both the project participants and interested observers. To understand how researchers benchmark community activity and analyse the outputs of various parts of your project, consider reviewing the work of the *Community Health Analytics in Open Source Software* (CHAOSS) Project.

3.10. Conclusion

The most effective way to achieve communication in open source projects is to show others kindness and courtesy, and to assume good intent upon first contact with people you've never met. Though this chapter contains any number of helpful best practices for effective communication, simply acting with graciousness to other people is the most important step one can take to communicate well. Remember there is a human being reading what you have written, and remember to treat them with the same respect you want for yourself.

[2] For a sample style guide, see PEP 8 — Style Guide for Python Code or the style guide for contributing to Mozilla Firefox, a project that employs multiple programming languages in its development.

[3] https://www.drupal.org/about accessed June 22, 2020 05:43 CET

[4] Kubernetes home page, https://kubernetes.io/, accessed June 22, 2020 05:57 CET

[5] The authors are grateful for the work of Kent C.Dodds and Sara Drasner in their article An Open Source Etiquette Guidebook, accessed 24 June 2020 12:52 CET.

[6] Gaia Vince, Evolution explains why we act differently online

[7] _https://producingoss.com/en/producingoss-letter.pdf, page 64, accessed 24 June 2020 11:46 CET

[8] *https://www.dreamwidth.org/legal/diversity* accessed 2 July 2020 13:37 CET (and how leet it is :)

Chapter 4. To Build Diverse Open Source Communities, Make Them Inclusive First

4.1. Introduction

Mere months after the COVID-19 pandemic forced most tech teams to work remotely or not at all, the murder of George Floyd ignited a racial reckoning in the United States. This movement hit the tech sector especially hard. While companies released statements in support of racial justice and declaring that Black Lives Matter, their own representation and experience of Black employees came under scrutiny, both internally and externally.

At first glance, open source seems immune to the multi-pronged problems with diversity and inclusion plaguing technology companies. Open source ecosystems operate online; contributors work across countries, languages, and timezones to power projects that drive technology forward. In some cases, people contribute purely out of love for these projects, not for free beer and stock. Given the prevalence of such altruistic intent, the thinking goes, open source is a true meritocracy, a space where all can thrive regardless of their pronouns or color of their skin. But if there's one thing that industry advocates love more than meritocracy, it's data. And when data show who contributes to the open source ecosystem, it paints a stark picture.

This chapter offers an overview of efforts by various open source communities to make projects more diverse and inclusive It also offers initial steps open source community managers and project maintainers can take to begin building communities and projects that benefit from a diverse contributor base. Most crucially, it argues that open source communities must practice inclusion before trying to find contributors from underrepresented groups. Without trying to fix the systemic barriers that prevent people from staying, efforts to diversify will keep failing in vain.

4.2. The state of open source diversity

In 2019, the U.S. Bureau of Labor Statistics found that 28.7% of total employed managers in the "Computer and Information Systems" sector were women. Of all managers in that sector, 9.6% were Black or African American, 15.8% were Asian, and 4.7% were Hispanic or Latino.

As this survey indicates, the managers in the technology industry in the United States are predominantly White and/or men. The open source communities skew even more towards contributors who are White and/or men. A 2017 GitHub survey of 6,000 open source users and developers found that 95% of randomly selected respondents were men. Three percent were women, and one percent were non-binary people.

How does this continue to happen? Recent years have seen an influx of diversity and inclusive (D&I) initiatives aimed at recruiting more diverse project contributors. The Linux Foundation has devoted an entire section of programming at its Open Source

Summits to sessions on D&I. DrupalCon offers scholarships for attendees and speakers from underrepresented backgrounds. Several mainstage keynotes at All Things Open 2019 discussed diversity. With so much attention from community leaders, why does contributor data still paint such a stark picture?

Relative lack of open source activity from women and underrepresented minorities (URMs) seems surprising at first, but a closer look reveals some persistent problems regarding inclusion in open source communities. Inclusion is often used as a synonym for diversity; in fact, it is its own principle. To quote Meg Bolger, "Inclusion is about folks with different identities feeling and/or being valued, leveraged, and welcomed within a given setting (e.g., your team, workplace, or industry)." There are some persistent, systemic barriers keep women and URMs from being included in open source communities. Without addressing and rectifying these systemic problems, diversity in open source won't improve.

Let's unpack three of those barriers: lack of free time to contribute, a hostile online environment, and lackluster documentation.

4.2.1. Lack of time to contribute for free

Globally, women continue to earn less than White men for performing the same professional roles. They are also increasingly likely to be their homes' primary earners, and they're more likely to do unpaid labor in the form of housework, childcare, and other domestic tasks.

As a result, they have less spare time to do even more unpaid work by contributing to open source communities. When they do make those contributions, they often do so during work hours—because they already work for organizations that let them do it.

Nisha Kumar (She/They) is the technical lead for container build, packaging, and distribution at the Open Source Technology Center at VMware. She also serves as co-maintainer for the Tern project, and a contributor to the SPDX and OCI communities. She juggles these projects with her role as primary caregiver to her two-year-old child. To Kumar, these dual roles are nothing new; as a girl growing up in India, she balanced schoolwork with chores, running errands, and caring for family members. These endless tasks left no time for her to pursue passion projects.

If not for her ability to work on open source during the day at VMWare, she's unsure if she could contribute today. As she told me:

> "As a woman in tech contributing to open source in my day job and getting compensated for it, I find that I still don't have the time to contribute to open source projects I am personally interested in or even volunteer for my local open source communities because I need to also take care of my child and household."

4.2.2. A hostile online environment

The internet is not an equally hospitable place. Women, people of color, and LBGTQ+ people are disproportionately targeted for online harassment, with women twice as likely to experience online sexual harassment as men. And these data were collected pre-2020.

The year's confluence of a global pandemic and mass civil unrest has led to increased attacks against people of color, specifically Asian Americans and Black Americans. If attackers feel emboldened to hurt people in the flesh, it's no surprise that this occurs online as well. If you work on open source projects with Black and Asian contributors, make no mistake: they are in pain.

"I don't see any acknowledgement that the internet, in general, is mostly inhabited by white men who are hostile to women and URMs," Kumar says. "How would anyone expect folks who are in general harassed on the internet to suddenly trust an open source community to not behave the same way towards them, despite what its Code of Conduct document says?"

Kumar's concerns are well-founded. Open source communities don't have a reputation for making most feel welcome. For years, project maintainers have written calls for help, saying they "feel drained, unappreciated, and even abused" on their worst days. A 2017 survey of 6,000 GitHub contributors found that 21% left projects they were working on after experiencing negative behavior. With the odds of such behavior that much higher for people from underrepresented groups, it's surprising they contribute to open source projects at all.

"Any time that I spend contributing to [open source communities] is time I could also be spending volunteering to advance LGBTQ equality, fighting against police brutality, and other issues that I care about, and that disproportionally affect underrepresented people—in fact, it's what makes us underrepresented in the first place," explains Perry Eising (He/They), who works as a community manager at a for-profit tech company that hosts open source projects. "It's challenging to justify to myself to spend time being involved in (sometimes hostile feeling) OS and tech debates when I have so many causes that require my attention and efforts."

4.2.3. Incomplete, inadequate documentation

The barriers above manifest in perhaps the biggest barrier of all: open source projects' lack of documentation. Quantitative data and qualitative interviews repeatedly share that most open source projects don't prioritize or reward documentation. As a result, outsiders often have no clue how to help.

I speak from personal experience here. After speaking at the Open Source Summit in Vancouver two years ago, I found the community so warm and welcoming that I resolved to find a project with which I could help. Given my background as a writer of all stripes, I thought writing documentation would be the perfect place to start and make a meaningful impact. So you can imagine how my face fell when I created my first GitHub account, logged onto the platform, and was left totally lost by where to go next. With no knowledge of how to fork, merge branches, or make pull requests, I had no clue where to start—and knew the margin for error was high.

I'm fortunate to live in a city with an active tech community that hosts regular meetups for techies of all interests, including tutorials aimed at beginners. I made my first pull request in person at a conference, and then made most of my first several dozen GitHub contributions at in-person events. Only after attending multiple free, accessible events over several months did I begin contributing independently. Absent any of these privileges, I would not have spent hours alone trying to learn how GitHub works to make a PR that might—or might not—get accepted.

The same GitHub survey that found one in five open source contributors leave projects due to bad behavior also found that incomplete or confusing documentation was the biggest challenge. 93% of that survey's respondents said they had encountered the problem, yet 60% said they rarely or never contributed to documentation. As a result, it's hard for project veterans to read their *own* projects' documentation.

Imagine how this makes outsiders feel.

"Contributing to open source is not only a technical challenge but equally a social challenge," explains Nuritzi Sanchez (She/Her), a senior open source program manager at GitLab. "Documentation is needed to enable remote asynchronous collaboration, which is how communities work. If everything is stuck inside of people's heads, that's going to create an atmosphere of confusion, frustration, and inefficiency." Documentation isn't just essential for code; it's also a guide to understanding each project's cultural and communication norms (refer to this guidebook's chapter on communication norms for more on this subject).

Open source communities use asynchronous communication to work cohesively across disparate time zones. Without clear documentation, prospective contributors won't know how decisions are made, where to contribute to the project, how teams collaborate, or why following certain processes is important. For prospective contributors who are non-native English speakers and/or have special needs, this lack of documentation makes contributing all but impossible. Inadequate documentation has far-reaching consequences. It shows a lack of transparency that wastes time, sows distrust, and prevents many open source communities from reaching their full potential.

4.3. Tips to build more inclusive projects and communities

Despite these barriers to entry, there's good news for maintainers: you hold enormous power to improve your project's culture by making it more inclusive. Community members, especially those from underrepresnted backgrounds, have discussed the lack of diversity and inclusion for years. Now, it's time for project maintainers to act by weaving inclusion throughout their project strategies—not making it an afterthought.

"[Diversity and inclusion] keynotes might have lofty ideals designed to raise awareness and some might even argue that they were useful at one point (maybe), but we've moved beyond that," argues Lisa-Marie Namphy [She/Her], who runs Cloud Native

Containers, the world's largest Cloud Native Computing Foundation (CNCF) user group. "Our communities are saying that it's time to act! And action means a change of policies, fund initiatives, [representation goals], so many things. The communities are asking for accountability, from the foundations who run them to the corporations who fund them."

If creating an inclusive community sounds overwhelming, remember that the community wants to help. If you're a project maintainer yourself, you don't have to do this work alone. In fact, taking the steps below with a trusted team will help improve your project for all.

4.3.1. Step one: stop saying you're a meritocracy

The first step to a more inclusive open source project involves understanding the meritocracy myth: the more you believe in meritocracy, the more biased your project is likely to be.

Why? Purely meritocratic projects don't acknowledge that people enter on unequal playing fields. If an open source maintainer isn't aware that women often have less time to contribute, or that LGBTQ+ contributors are more likely to endure online abuse, they won't take steps to make the community more inclusive. As a result, they risk losing the diverse contributors they worked hard to recruit.

Terri Oda (She/Her) volunteers for the Python Software Foundation and Google's Summer of Code, alongside her role as an open source security researcher at Intel. She says claims of meritocracy make her cringe. Why? Such statements cause maintainers to ignore opportunities to get more people involved in projects, even in cases where the open source community gathers in person.

"For example, say you're running code sprints at a conference and want to increase the number of women," Oda says. "If you're thinking about merit and skills, you're going to wind up offering more intro-to-sprinting type content. But if you look at the bigger picture, you might realize that the conference offers childcare during the main conference, but it stops when sprints start. Or that the venue isn't in a safe area and the sprints run until after dark."

The first step to build a more inclusive environment is self-awareness. Open source contributors enter projects with a range of lived experiences that affect how—and if—they show up. Sitting with and reflecting on this fact is the first, most crucial step.

The next step is to take an honest look at your project's current community, and take note of who is—and isn't—there. If your project contributors all, or even mostly, look like you, that's a huge red flag that an inclusive overhaul is in order.

4.3.2. Step two: prioritize your project's documentation

A 2019 Stack Overflow study found that about 41% of developers have less than five years of experience. Between these new technologists and current emphasis on STEM education, there are lots of opportunities to welcome new open source contributors. In order to do so, project maintainers must lower barriers to entry—and clear, concise documentation is the first step.

Zach Corleissen (He/They) is the lead technical writer for The Linux Foundation (LF) who recently revised a large architectural document for the LF Energy Foundation. He also serves as one of the co-chairs for the Kubernetes documentation special interest group (SIG Docs). Kubernetes was his first open source software project, and it quickly became one of the most prolific projects in modern open source. Its rapid growth allowed Corleissen to own important aspects of its documentation, and revise it to become more reader-friendly.

"Insisting that code is self-documenting is a form of gatekeeping [and] an example of an unhealthy project culture," Corleissen says. "I think the devaluation often comes from developers who see a static generator stack and think, 'How hard can it be?' One of my least favorite dismissive phrases: 'It's just a pile of Markdown.' If only it were that easy! Documentation is code for an environment where no chipsets are identical; kernel defaults are hostile; RAM is variable; storage is subject to random external dependencies; and production regularly fails despite optimal conditions, or inversely, succeeds in spite of obvious CI failures."

To track progress, the SIG Docs group does a quarterly review where they measure the progress of their previous quarter's goals and prioritize work for the upcoming quarter. One of their community rules centers on ownership: in order to adopt a goal, a project needs a specific person willing to drive it.

4.3.3. Step three: create and enforce a clear code of conduct

If your project doesn't already have a code of conduct (CoC), it's never too late to make one (refer to this guidebook's chapter on governance for tips of getting started). They are an expectation for modern open source initiatives, from long-term projects to two-day conferences.

In my own research for this chapter, several open source contributors told me they won't consider joining new projects that lack clear CoCs; for these URMs, the risk of joining an unwelcoming if not hostile community is too high. "Having a code of conduct would be big for me," explains Natalie Zamani (She/Her), Senior Software Engineer at Apple. "And then something as seemingly unrelated as not tolerating project contributors espousing racist/sexist/homophobic/transphobic ideas, even if it's not related to their project work. I wouldn't feel comfortable working with individuals who hold such views, full stop. And I've seen a few projects that would otherwise be interesting to me where that's tolerated."

As the former President and Chairperson of the Board of Directors for the GNOME Foundation, Sanchez helped create GNOME's event CoC. She says that while the Contributor's Covenant is the default code of conduct for a lot of open source communities, translating it to an events format took some creative work—and a lot of feedback from the GNOME community.

"No matter the type of CoC you're rolling out, having a transparent plan and timeline is key," Sanchez says. "At GNOME, we created a working group after one of our annual conferences to start drafting a code of conduct. We passed the notion of a working group by the Board of Directors to make sure that they were on board. They made a community-wide announcement letting people know the process: a working group would be drafting the CoC, sending to community for revisions, the Board would then see the revised draft and vote, and then the membership would vote at the Annual General Meeting."

Despite the key role of community feedback, Sanchez says the CoC should be owned by a governing body within your project. CoCs remain a touchy subject in open source communities, and not all open source contributors believe they're necessary. A governing body (or at least a committee) comprised of diverse contributors that shares the creation process can help alleviate disagreements. Once you've created your governing body, assign members to own specific tasks. These include a Chair who can break voter ties, moderators to enforce the code of conduct, and mentors to train the community. It's essential for all community members—especially URMs—to see their safety and integrity is protected by project leadership.

"I am a firm believer that signalling is very important, but that broken trust is difficult to repair," Eising explains. "Don't signal to [underrepresented people] that you are ready to embrace them before you actually are—that's like inviting someone who uses a wheelchair to a party on [an upper] floor with no elevator. That person won't trust you again to think about their needs appropriately. Organizations need to look within and really assess before making a reach out."

4.3.4. Step four: reward contributions beyond code

In her time working on open source, Sanchez says that most projects focus on attracting contributors to a narrow set of project work: engineering, design, translation, documentation, and outreach. Despite how broad that sounds, she's offered the table below that reflects many more roles and types of contributions she'd like to see rewarded:

`Around making personal career development goals, you can look into connecting a type of skillset with specific teams within open source software organizations.

Table 1. Aligning project role to career goals and skillsets

Career development target	**Teams within OSS orgs to check out**	**Why**
Sales and business development	Fundraising, partnerships	Both of these things require you to pitch the value of the open source community / project and require you to develop your communication and negotiation skills, among other things

Career development target	**Teams within OSS orgs to check out**	**Why**
Marketing skills	Engagement, marketing, or outreach teams	Some projects may not even have this set up and are in need of someone to help. Even if you don't have a lot of experience in this, you may have more experience than anyone else in that community and it's your chance to build something from scratch. This could look really amazing on a resume!
Strategy skills	Board of directors / governance team, community team	It depends a bit on the maturity of the organization, but typically there's a lot of room for building your strategy skills when on a board of directors. You have a birds-eye view of the project, typically have say over project finances, and can help define goals and move the project forward at a whole new level. Since you can't get there right away, leading initiatives can help you build those skills and there's often a lot of room for people to step in and own big chunks on open source community teams
Data science skills	Community team, board of directors	What kind of data is being collected to ensure that initiatives are successful? Measuring a community's health is something that more and more people are interested in and there's a need for those interested in data analysis to help

Career development target	Teams within OSS orgs to check out	Why
Graphic design skills	Marketing team, technical projects	There's a lot of need for graphic design for brand and marketing initiatives, and in general to help make the project more mature. Some projects may not even have established brand guidelines, and there's a big need for more designers in general
Project management and program management skills	Engagement, marketing, outreach, documentation, community teams	There is a huge need for highly organized people who can create processes and structure. Many initiatives fall to the side because there isn't someone to help push it along and make it happen
Product management skills	Any technical project, new initiatives, website, newcomers initiatives	Product managers (PM) are essential at companies, and yet it's something that isn't always easily found within open source software. There's a lot of room for PMs to jump in to help create more innovative products and help bridge the gap between communities and businesses, helping to expand the project's reach
Legal skills	Board of directors or community team	There's a growing need for more people who are able to navigate open source related legal matters. Lawyers may get a lot of great experience working on community teams or sitting on the Board of Directors
HR/people skills	Board of directors, community team, newcomers initiatives	We need people who care about people and want to make the community awesome.

This list isn't exhaustive, nor is it applicable to all projects. For more on this topic, refer to this guidebook's chapter on the range of roles in open source projects. The goal is to look at your own open source project's holistic needs in the short and long terms, then recruit contributors to fill specific gaps. Doing so allows you to create a governing board with representatives that own specific aspects of the project and contribute to its growth.

Nithya Ruff (She/Her) leads the Open Source Program Office for Comcast and serves as Chair of the Linux Foundation board. In more than two decades of open source work, she has seen how ignoring crucial skills—including legal issues such as copyrights and trademarks—can keep a project from achieving long-term success. Recruiting and rewarding diverse contributions also plays a key role in preventing burnout, which project maintainers have been increasingly vocal about.

"It is unfair to expect the maintainer or the developer who started the project or leads the project to care for all of these issues, [or] have the skills to do it," Ruff says. "All forms of contribution need to be valued [because this] brings diversity of people into the project, which makes the project more vibrant and innovative. Foundations like the Apache Software Foundation [and] Linux Foundation bring all of these contributions to the table for their hosted projects. This allows the project to more successfully build a broader ecosystem."

4.3.5. Step five: mentor new talent to grow and lead the project

Eleven years after co-founding Redis, Salvatore Sanfilippo announced plans to step down as project maintainer of the NoSQL database. He named Yossi Gottlieb and Oran Agra as his successors to maintain the Redis project. In doing so, the Redis governance model got a refresh.

Rather than keeping Redis's prior BDFL style, Gottlieb and Agra built a new, lighter governance model. It involves electing a small group of longtime Redis developers to act as core contributors and uphold the project's Code of Conduct.

Regardless of your own project's governance model, you must include a way to train key contributors to assume leadership roles. This achieves three key goals:

1. Helping new contributors learn how they can grow
2. Rewarding contributors who own key aspects of the project
3. Preventing maintainer burnout

This last point is noteworthy: Sanfilippo said when he stepped down from Redis that despite his passion for coding, he never aspired to maintain a project. Without new leaders to step up—and documentation sharing how contributors can assume such roles—maintainers risk either working on projects when they no longer want to or having the project stall. Likewise, the project risks missing an opportunity to give interested contributors a chance to step up.

The act of building and maintaining a mentorship program is inclusive in itself. Several open source leaders interviewed for this book said they see a clear need for more mentorship in open source at large, and a desire to do it themselves. In some cases, open source contributors believe so much in the power of mentorship that they restructured their contributions to include it. And, because they were mindful of their own time limitations, they offered flexibility to new leaders as well.

"My open source contributions definitely changed even before I became a parent," explains Oda. "As the coordinator for a global mentoring program that happens in the summer, I had to plan some years ahead to build a volunteer team that could do everything I do. So, I handed off some of my other projects more completely and never went back to them.

"Since new moms typically get less than one hour of free time per day, the key for me has been aligning the open source I want to do with the open source that work wanted to pay me for. I worked to take [the] CVE Binary Tool open source after I returned from maternity leave, and worked with my boss to make sure I could have time to mentor students as part of my maintainer role."

To build your own mentorship program, Sanchez says to focus on four actions and initiatives:

> Create learning opportunities often. Find ways to help people learn what you do and how you do it. Don’t just wait for formal internship or mentorship programs, but take advantage of those if you can. Consider recording videos, holding AMAs, participating in events, etc. Be open to communicating with people informally in order to build relationships and trust so that you can help develop those with potential. Cast your net wide and you’ll probably find those gem contributors who are ready to step in to help bring your project to a whole new level.
>
> Be a connector. Try to have a mental map of prominent contributors in your community and their strengths. Share the mentorship by introducing newcomers to several people. Burnout is real on the mentor side and you want to make sure that there are other people your mentee can reach if you need to take a break or just get busy.
>
> Make sure that there’s a chat tool specifically for community interactions. In order to build trust, people need private spaces. Chat facilitates conversations and collaboration, and also allows people to message you directly. To avoid burnout, you may want to have a chat tool available just for your community / work conversations and a chat tool just for your personal life. That way, you can turn off all notifications on one tool if you need a break, or just simply have that mental separation thanks to differences in UX.
>
> Connect through events. Events provide a powerful opportunity for you to connect with potential mentees. At these events, try to plan fun activities that are designed to help people connect informally. This may mean having a people-bingo where people have to ask each other questions to enter a raffle, or it could be a city tour, or a game night. Fun activities throughout the year can facilitate authentic relationships, which can also help people overcome fear of contribution.

For more ideas around mentoring in open source communities, refer to this guidebook’s chapter on building a culture of mentorship.

4.3.6. Step six: commit to continuous improvement

The work of inclusion is never done. It’s ongoing. As your project grows, you will find new gaps to fill, questions to document, and additions to the code of conduct. As your community becomes more inclusive, it might feel like you’re finding more ways you’ve fallen short. Uncomfortable as this is, it’s actually a good thing. It means you’ve done the hard work of committing to keep on getting better. And, if you’ve done the work of building an inclusive team, you won’t do this work alone. Instead, you’ll share the work with your community, giving everyone the chance to share their feedback.

To keep the dialogue ongoing and open, give your community options to leave feedback on their experiences. This can range from quarterly surveys to giving contributors the freedom to create channels in your project’s communication platforms to chat about mental health, being a person of color, how to handle negotiations, etc. Such channels give contributors ways to connect socially, which is crucial for increasing asynchronous collaboration. It also gives you new ways to support contributors so they can contribute more fully.

"I am hearing-impaired," explains Don Watkins (He/Him), a community correspondent for OpenSource.com who has been active in the Linux community for two decades. "I was particularly impressed when attending the Creative Commons Global Summit in Toronto in 2018, where nearly all presentations were accompanied by folks who signed and also provide simultaneous closed captioning of all speakers."

Inclusion isn’t a one-time pull request, but an ongoing, important activity. Without building and sustaining inclusive communities, there’s no hope of improving diversity of open source contributors. To recruit new talent, prevent maintainer burnout, and create affirming online environments, open source maintainers should commit to inclusion. Change starts from within, and when technology contributors from a variety of backgrounds see your inclusive efforts, they will be much more likely to join.

"Make it easy for people to get involved and to contribute back," says Ruff. "The mark of a good project is not how complex it is, but how easy it is to get involved."

Chapter 5. Why Do People Participate in Open Source Communities?

5.1. Introduction

Why do people participate in open source communities? What do they get out of it? Are they focused on their own needs, or are they mostly thinking about others?

Given that people are involved, the answer—as you might guess—is complicated and varied.

One need only consider the behaviors we see in open source communities to very quickly intuit that there’s no single force at work here. After all, we see plenty of contributors on most big open source projects who are doing so at part of their day job. That doesn’t mean they don’t care about open source beyond a paycheck. But it does suggest different reasons for participation than the contributor working nights and weekends on their own passion project. Furthermore, we see contributors who are focused on solving some interesting technical problems while others are explicitly trying to bring about a societal benefit.

But that doesn’t mean we can’t suss out some common patterns. Not only has there been academic research into open course contributions specifically, but there’s a whole body of psychology literature dealing with motivation. As a result, motivation is the lens we’ll use in this chapter to examine the question of why contributors participate in open source.

5.2. Extrinsic motivation

We could say that this type of motivator is about "Show me the money!" and leave it at that. Extrinsic motivation isn’t quite that simple, but payment in whatever form that took during a given era and place has historically been the go-to way to get people to do something. Whether to accumulate riches or provide themselves with basic necessities such as food and shelter. The idea that you do work you otherwise wouldn’t bother with in exchange for something of value is deep-rooted in most human societies.

However obvious the existence of extrinsic motivation may seem to the average person, it wasn’t really a field of formal study until the 1940s when behaviorist Clark Hull, later working with Kenneth Spence, came up with the idea of drive reduction theory. This theory focused on reducing primary drives like hunger and thirst. If you’re hungry you eat. If you’re thirsty you drink. Drive reduction theory fell out of fashion, in part because it *didn’t* focus on things—like money—that could be used to reduce drives but only in a secondary way. However, we see echoes of drive reduction theory in familiar concepts like Maslow’s hierarchy of needs.

Coming back to what makes contributors work on open source, money can, of course, be a big carrot. Even going back twenty years or so, many major open source projects had a significant number of contributors who were paid by their companies to work on open source.

Career advancement goes hand in hand with pay. But is open source software any different from proprietary software development in this regard? It may be. Multiple researchers have found empirical evidence to support the idea that there's at least a perceived advantage to support the argument that there are career advantages to developing code that's in the open and therefore available for others to look at and work with.

More recently, there's been significant empirical evidence to support the argument that there are career advantages to developing code and contributing content that's in the open and therefore available for others to look at and work with. Indeed, the idea of "GitHub-as-resume" has become a common (if sometimes inappropriately applied) theme.

5.3. Intrinsic motivation

Research beginning in the 1970s started to focus on intrinsic motivations, which do not require an apparent reward other than the activity itself. Self-Determination Theory, developed by Edward Deci and Richard Ryan, would later evolve from studies comparing intrinsic and extrinsic motives, and from a growing understanding of the dominant role intrinsic motivations can play in behavior.

With respect to open source software, it's perhaps the ideological or altruistic motivations that first come to mind here. After all, in the case of free software at least, there was always an ideological and political element from the beginning even if there were also practical benefits for a user to have access to source code.

There's some evidence from surveys that contributors can be somewhat motivated by these factors. However, the research results are mixed. Some contributors say on surveys that they're participating, in part, for ideological or altruistic reasons. But the effect is most pronounced among contributors doing open source as a hobby. And while altruism certainly can influence professional contributors, this mostly seems to be the case among contributors who are *also* satisfied with their pay and other aspects of their career.

A variant of altruism is kinship amity. It's related to the concept of gift economies but is specific to family (kin) groups which don't expect a calculated *quid pro quo*. (In other words, family members do things for each other but they don't typically keep a running tally, at least a systematic one, of who hasn't been pulling their weight recently.) It's different from altruism in that it is restricted to the group to which one belongs, such as an open source community. Here again the research is mixed. Studies have generally found a positive relationship between identified kinship amity and various measurements of effort, such as number of hours worked per week—but the correlation is often fairly weak.

Finally, there's just fun. This should come as no surprise to anyone who hangs around open source developers, designers, and content creators. Most of them *like* working on open source projects. One large study from 2007 (Luthiger and Jungwirth) determined that fun accounted for 28 percent of the effort (hours) dedicated to projects. Though, again, it’s easier to feel motivated by fun and altruism when it’s either a hobby or you’re otherwise satisfied professionally.

5.4. Internalized extrinsic motivation

Today’s psychology literature also includes the idea of internalized extrinsic motivations. These are extrinsic motivations such as gaining skills to enhance career opportunities—but they’ve been internalized so that the motivation comes from within rather than coming as a direct result of a carrot being dangled by someone else. It’s the difference between learning a new language because you know keeping current on new tech pays off over time versus receiving a salary increase as a direct payoff for earning a certification. Gaining a good peer reputation, among community insiders and potential employers, for work in open source is one good example of how this type of motivation can play out in open source development. A variety of surveys have supported the idea that peer reputation is a driver for participation.

Learning is also frequently mentioned as a big benefit to participating in open source projects. Learning may be intrinsically satisfying but it can also be an important ingredient of career advancement. It can be hard to tease apart the intrinsic from the extrinsic here though. Is it learning for learning’s sake? Is it learning for specific skills needed on the job?

A final motivator in this category is what researchers call "own-use value" but is more recognizably described as something like "scratch your own itch." Develop something that you want for yourself and create something for others in the process. That something is ultimately an external reward to yourself but no one is forcing you to do it.

5.5. Conclusion

As noted earlier, motivations for many things are multi-faceted and contributing to open source software is no exception. However, here are three takeaways that you may find useful.

Don’t expect non-extrinsic motivators to carry too much of the load. Many do contribute to open source projects for idealistic or altruistic reasons. But those are usually not the sole motivators and may not even be important ones. Especially in the case of projects that have commercial backing, pay, and other professional working benefits matter a great deal.

Ampily non-extrinsic motivators such as learning and peer recognition. While not replacements for more direct benefits, the opportunity to be recognized by peers and to work in new technology areas are motivators. Organizations should consider peer recognition programs and explicitly encourage learning in order to make the best use of these motivational factors.

Motivators can be counter-productive when over-rotated. Precisely because motivations are multi-faceted, don’t place too big a bet on any single one. We already discussed the limitations of ideology and altruism. However, also consider something like "scratch your own itch" for example. If that’s the *only* reason someone contributes, they’ll tend to wander in and out of a project as their own specific needs merit. That may be a perfectly fine outcome, but it’s not a path to developing a long-term maintainer.

Chapter 6. Project and Community Governance

6.1. Introduction

In this chapter, we'll discuss assessing and evolving an open source project or community governance model.

All organizations operate in and with governance structures. The term "governance" carries multiple meanings in an organizational context. It can refer to regulatory matters or risk management issues, for example. More generally, though, it can also refer to a system of rules, roles, and procedures that determine how power in an organization gets distributed.

Because open source projects are organizations, every one features governance structures. Some of these structures are more *explicit* than others. Some are more *formal* than others. But every project has them.

Unfortunately, too many discussions of open source project governance focus on activities or resources, like "speaking for the project" or "ownership of the web domain." While documenting these functions is useful, we should remember that these are *aspects* of a project's governance, but they are not the full extent of it. At its heart, open source project and community governance is about *people*—their rights and responsibilities as part of a project and the expectations others have for them.

6.2. What is governance?

Simply put, "governance" refers to, "The rules or customs that determine who gets to do what (or is supposed to do what), how they're supposed to do it, and when."

Two categories of governance-related issues are most pertinent to open source projects: those related to *roles* and those related to *policies and procedures*. For the purpose of explanation, we'll discuss each of these issues separately. In practice, however, they're inseparable—two sides of the same coin, as our forthcoming example will demonstrate.

6.2.1. Roles

A great deal of activity hinges on roles-related governance in open source projects. Think of a *role* as a function someone in the project performs.

When analyzing your own project's roles-related governance systems, ask the following questions:

- What roles do (or can) project contributors play?
- What qualifies a person to play a particular role in the project?
- What duties, privileges, and forms of authority are associated with each role?

- What project resources are the province or responsibility of people who perform certain roles?

6.2.2. Policies and procedures

While contributors occupy certain roles in an open source project, the project's governance structure also determines how those people do what they do as part of the project. *Policies and procedures* refer to the processes and guidelines contributors follow when performing their roles.

When analyzing your project's procedure- or policy-related governance systems, ask the following questions:

- How do various contributions get accepted into the project?
- How do contributors come to occupy and eventually depart from certain roles in the project?
- How can role descriptions and responsibilities be changed?
- How do project decisions get made (and by whom)?
- How are debates and conflicts resolved (and by whom)?

6.2.3. Roles, policies, procedures: an example

Remember: A project's roles and its policies and procedures aren't discrete; they're interwoven as part of the project's overall governance model. For example, consider a project role called "Documentation Maintainer." A project might outline that role this way:

1. Role Title: Documentation Maintainer
2. Qualifications: several years of consistent contributions to documentation
3. Access to Role: other Documentation Maintainers can nominate and vote to grant this role to eligible contributors
4. Duties: write documentation and review documentation from other contributors
5. Privileges: speak for the documentation team, participate in development meetings
6. Authority: ultimate decision on documentation content, technology, and strategy
7. Change Procedure: other Documentation Maintainers vote on changes to role description

In many cases, a project's actual role descriptions are more elaborate than that (some projects' role handbooks are dozens of pages long). As projects mature, the number of different roles people play in them can increase. Moreover, larger and more mature projects associate roles with *collectives*—that is, a group of contributors can perform a certain role jointly. For instance, a project might feature several "steering committees," each with its own set of election procedures. This is true for the Kubernetes project, where "special interest groups" are a popular unit of governance. In that project, the role of Code Contributor is subdivided further by interest group (team) and by contributor level (Member, Reviewer, Approver, and Owner). So a contributor's *actual* role would be something like "SIG–Network Approver," not just "Code Contributor."

6.3. Why governance?

In some open source communities, the idea of "governance" has a poor reputation. This is true in cases where project contributors tend to construe governance as a purely negative force—a set of rules or procedures aimed solely at telling people what they can't do, how they shouldn't act, or how they should limit themselves to acting only within certain boundaries.

But a well-crafted governance model can in fact be a largely positive force in open source communities. A project's governance model outlines that project's *terms of engagement*—the specific, tried-and-tested structures for working together and making decisions that project contributors have found works best for the community. A clear governance model encourages new contributors to become involved in your project.

A well-designed system of governance is much less likely to turn away or de-motivate project participants than a vague or non-existent one is. Consider your project from the perspective of new contributors. Are new contributors *more* or *less* likely to jump into a project without any sense of the role they're supposed to play and the rules they're supposed to follow when they want others to seriously consider your contributions? A clear governance model helps people understand precisely how they can make an immediate contribution to a project, how they can pitch in without upsetting the project's rhythms, how they can escalate questions or issues if they have them, and what sorts of leadership positions they can aspire to if they stick around long enough. So a community's goal in architecting a governance model should be, "Make structures of participation obvious." When your project's rules are clear, contributors can engage with confidence. Taking this approach to governance can positively impact a project's long-term viability and growth.

6.4. Making governance explicit

Recall that every open source project features different roles contributors can play and procedures that people in those roles are typically (or should be) following. This is true whether or not those roles and procedures are actually documented anywhere. Where roles aren't documented, they are implicit in regular contributors' activities (and not infrequently a source of argument). Successful open source projects make their governance models explicit.

In a 2018 study, researcher Javier Canovas found that 19 of the 25 most-starred projects on GitHub hadn't published documents outlining their governance models. Canovas considered this unfortunate for several reasons.

"First, [explicit governance] helps promote an organization's sense of transparency," he writes. "One could know how much time a group takes to consider an issue, the chances contributions have of making an impact on the organization, or who is going to hear their voices when they speak up. Second, explicitly defining a governance model may also help one better understand and classify how open organizations are driven."

Here’s an example of how this works: in 2018, the Kubernetes project added a set of detailed, comprehensive Role Handbooks for their Release Team. These handbooks outlined information related to the Release Team role, including qualifications necessary for joining the team, duties members of the team perform, and details on the team’s decision-making processes. As a result, the Release Team became the most popular point of entry for project contributions; new participants knew exactly what to expect. Other teams within Kubernetes followed suit—and experienced a doubling or even tripling of the number of new contributors.

Clear and explicit governance models have another critical benefit—cultivating a strong sense of trust in your project’s community. Members of projects with robust, detailed governance models benefit from a shared commitment to a transparent set of procedures, policies, and role descriptions. They can appeal to a commonly understood set of guidelines when disputes arise. All of this makes questions about participants' motives, intentions, goals, and authority less contentious.

6.5. How community-originated projects evolve

Open source projects rarely begin by "selecting" and implementing a perfectly preconceived governance model. Much more commonly, projects' governance models evolve as their communities grow and diversify.

In its early days, a project might only have one or two developers, making discussions of "governance" largely irrelevant (the project is simply not big enough to have a need for any structured decision-making process). But this will change as the project attracts additional contributors. And because a project’s governance model, its culture, and the behaviors of its leaders are all intimately entwined, any change to one will likely spur changes in the others. While every project is different—growing in its own way and following its own trajectory of maturation—we might note certain common, recurring milestones in a project’s development that tend to trigger governance evolutions.

6.5.1. Work among founders (1 or 2 members)

Projects that start with a single developer (or small group of developers) do not often require any formal governance structure. Gauging consensus is easy, and during the early stages of a project, disagreements about what should be done (and who should do it) are rare. A project’s early members all typically have carte blanche to take the actions they see as best for the project, like approving code for inclusion. Normally, no structure is required in addition to a GitHub repository, and all early developers receive project membership status almost immediately.

6.5.2. Early project growth (up to 5 members)

As projects begin growing, the limitations of this approach become obvious. When a project has even five developers, coordinating work becomes more difficult, and newer developers may not be immediately familiar with the design choices and coding standards the project's early developers have followed.

So the first evolution projects tend to undergo is often one that requires code submissions to undergo peer review before being merged. The "first level" of the project's hierarchy consists of those with the authority to approve pull requests or code and content submissions for inclusion in the project. Initially, deciding who receives this authority is easy; the project's original, trusted developers all receive it, and the project founder acts as final arbiter in case of disagreements.

6.5.3. Mid-term project growth (10 to 15 members)

The next event to trigger a project governance evolution is often related to how people who join the project become recognized members of the group. This tends to occur when the size of the project has increased to approximately 10 or 15 developers. At this point, a project community typically must develop more formal guidelines for admitting new project members.

One common standard projects use to assess new members is sustained participation (how long and how often the contributor has been active in the project) combined with a judgment about what one might call "good taste"—an assessment about the quality of work a contributor tends to submit, that contributor's good judgement in review comments, etc. Still, the project founder tends to be the gatekeeper and final arbiter of who gets promoted inside the project.

6.6. How corporate-originated projects evolve

Some open source projects that begin life as the work of a professional software development team operating in a corporate environment tend to evolve somewhat differently. Because these projects originate in corporate environments, they often inherit the organizational structure of those environments. They may, for example, already feature a robust group of developers with their own notions of hierarchy (managers, architects, junior and senior developers, and so on).

6.6.1. Early-stage corporate-originated projects

Initial efforts to increase community engagement in the projects tends to focus on growing adoption and engaging with early users. Pre-existing developer teams typically continue project planning, however, in a centralized manner. For this reason, external contributors may find engaging with the project more difficult—and the project may not gain sufficient traction as a result. The rapid pace of project changes, the opacity of the planning process, and the strength of pre-existing relationships between the project's developers can make feature development more difficult for external contributors. Early patch submissions may stay unreviewed for longer periods of time, and these submissions will be relatively infrequent.

This is as far as many corporate-originated projects will evolve. While the core team may engage actively with the project's user base, resources required to *grow* that developer base are considerable, and many organizations choose not to make the investment.

However, one oft-cited benefit of the open source model is an ability to collaborate with industry partners and competitors and share the burden of development of common requirements. If this is a goal, then growing participation in a corporate-originated project beyond a single vendor is critical.

6.6.2. Evolving to multi-vendor corporate open source

For corporate-originated projects, expanding project participation involves engaging with both interested individuals who are using the project and vendors who might be motivated to invest in the project. Uniting these parties will have implications for project governance.

Many projects begin enticing other vendors to contribute by demonstrating a viable market for the project. Vendors typically do not invest sustainably in open source projects unless they can justify that investment. Illustrating significant and enthusiastic user adoption of the software is therefore critical at this stage. Initial efforts focus on accelerating adoption momentum and successfully converting users into contributors by soliciting their active participation in the project roadmap and project promotion.

Alternatively, a project may attempt to engage with other vendors by focusing on encouraging collaborators to "build on" a common platform. While companies may not be able to justify significant investment in the project "core," they may be able to justify investment in *extensions* to a project—if those extensions are relatively inexpensive and can support their business.

For example, by focusing initial outreach and engagement efforts on the APIs, the developer experience for extensions, and the path to distribution for people writing those extensions, projects may grow large communities of vendors building atop a platform, rather than modifying the core platform itself. Distinguishing these two areas of development—between the "core" and the "periphery"—often involves making governance decisions specific to each (only some project roles may receive permission to operate in the project "core," for instance).

When a corporate-originated project has demonstrated substantial market opportunity (either by proving that the project fills a significant gap in the market or by growing a large user base directly), it can engage with potential vendor partners to collaborate on the project. This discussion is partly technical and partly business-focused.

Before making a significant investment of engineering resources in a project, vendors will likely ask:

- Can we engage with the project on a level playing field? Or do stakeholders use different processes to evaluate changes from different vendors (Contributor Licensing Agreements that give additional rights to the originating vendor over others, for example)? One common way to ensure a level playing field from a legal perspective is to contribute the project’s management and trademark to a foundation.
- Does this project meet a customer need? Vendors will consider market fit, and how

the project fits into their product portfolio.

Accepting participation from additional vendors can significantly impact a project's governance. One way to ease potentially turbulent impacts is to target vendors with whom the originating vendor does not compete directly. For example, a cloud hosting company may have more success recruiting a vendor of on-premise software products to its project than it would recruiting a competing hosting vendor. Competing vendors may only be willing to join when a project can demonstrate a consistent record of multi-vendor engagement in the project.

6.7. Governing sustained evolution

Once project participation reaches a kind of "critical mass," many common patterns emerge—regardless of whether an individual or corporation has initiated a project.

In all the cases we've discussed so far, rules and procedures for decision making tend to be implicit. And since most open source projects never recruit more than 10 active developers (or one core vendor), most projects never reach a point where explicitly documenting project governance becomes necessary. Those that do, however, will likely adopt even more nuanced and complex governance models. Refer to "Examples of open source governance models" below to learn more about these.

Sometimes, when projects reach this size, they seek to transition management and trademark of a project to an independent entity (usually called "foundations" in the open source world). On rare occasions, projects may establish their own independent consortium for this purpose. More frequently, however, a project will approach an existing foundation (such as the Apache Software Foundation, the Linux Foundation, the Cloud Native Computing Foundation, the Eclipse Foundation, the OpenStack Foundation, or the Software Freedom Conservancy, to name just a few) and ask the foundation to adopt the project.

When selecting a foundation with whom to partner in this way, open source projects must make several considerations, including:

1. Cost structure;
2. Governance requirements imposed by the foundation;
3. Affinity of the foundation with the user and developer base of the project.

At this point, projects will commonly discuss the extent to which member fees should influence the project's technical governance. Two dominant models for this governance exist.

The first is a strict separation of funding and technical inputs, where the members who join at the highest membership level have input into (and can influence) project budgetary matters (for example, how funds will be disbursed between infrastructure, headcount, marketing, events), but technical merit dictates how the project is governed technically. The second is a "pure member" organization, where members are entitled to appoint representatives to a technical governing board with oversight on which sub-projects will be adopted in the project, and how the projects will be governed.

Foundations can play another key role in a project's evolution: defining the market dynamics around the project, including administration of the project trademark. A trademark is one of an open source project's most valuable resources for guaranteeing that vendors are distributing the project (or derivatives of it) in a way that does not damage the project's reputation. Open source projects commonly use trademark certification as a way to "bless" certain vendor products in the market or to influence the way derivative products behave.

Some projects hold tightly to the idea that contributors are *individual contributors* and not representatives of companies for which they may happen to work. In mature open source projects (like the Linux kernel), this allows people to maintain community status and seniority even when they change employers.

6.8. Examples of open source project governance models

6.8.1. "Do-ocracy"

Open source projects adopting the "do-ocracy" governance model tend to forgo formal and elaborate governance conventions and instead insist that "decisions are made by those who do the work." In other words: In a do-ocracy, members gain authority by making the most consistent contributions. Peer review remains common under this model; however, individual contributors tend to retain de facto decision-making power over project components on which they've worked most closely.

For this reason, some do-ocracies will claim to have "no governance at all," relying instead on individual stakeholders' authority to make decisions on matters "where they've done the most work." But as we've already explained, such claims about an absence of governance are misguided. Every open source project has a governance model. In the case of most do-ocracies, the governance model is merely implicit in the everyday interactions of project members. As a result, joining them can be difficult and intimidating for newcomers, as would-be contributors might not immediately know how to participate or seek approval for their contributions.

To get started in a project with this governance model: Find an aspect of the project you feel you can improve and simply begin working. Review the recorded history of changes to the project to identify the participants whose feedback will be integral to your successful contribution. As the project accepts more of your contributions, you will gradually accrue influence in the community. Do not expect to influence decisions in a do-ocracy until you are able to demonstrate a history of successful contribution.

6.8.2. Founder-leader

The founder-leader governance model is most common among new projects or those with a small number of contributors (and since most open source projects have only a small number of contributors, this is a rather popular model!). In these projects, the individual or group who started the project also administers the project, establishes its vision, controls all permissions to merge code into it, and assumes the right to speak for it in public. Some projects refer to their founder-leaders as "BDFLs" or "Benevolent Dictators for Life," a term that is falling out of fashion.

In projects following the founder-leader model, lines of power and authority are typically quite clear; they radiate from founder-leaders, who are the final decision-makers for all project matters. This model’s limitations become apparent as a project grows to a certain size. Separating the founder-leaders' personal preferences from project design decisions eventually becomes difficult, and founder-leaders can become bottlenecks for project decision-making work. In extreme cases, founder-leader models can create a kind of "caste" system in a project, as non-founders begin feeling like they’re unable to affect changes that aren’t in line with a founder’s vision. Disagreements can lead to project splits. Worse, a founder-leader’s disappearance, whether due to burnout or planned retirement, can cause a project to disintegrate entirely.

To get started in a project with this governance model: Browse project mailing lists or discussion forums to identify the project’s founder-leaders, then address questions about participation and contribution to those leaders through one of the community’s public communication channels. Founder-leaders tend to have a comprehensive view of the project’s needs and will direct you to areas of the project that will benefit most from your contribution. Be sure to understand founder-leaders' vision for the project, as most founder-leaders will veto proposed changes they feel conflict with that vision. When starting out, do not expect to propose changes that will not serve the founder-leaders' vision for the project.

6.8.3. Self-appointing council or board

Recognizing shortcomings of the founder-leader model, the self-appointing council or board model aims to better facilitate community leadership turnover and succession. Under this model, members of an open source project may appoint a number of leadership groups to govern various aspects of a project. Such groups may have names like "steering committee," "committer council," "technical operating committee, "architecture council," or "board of directors." And typically, these groups construct their own decision-making conventions and succession procedures.

The self-appointing council or board governance model is useful in cases where a project does not have a sponsoring foundation and establishing electoral mechanisms is prohibitively difficult. But the model’s drawbacks become apparent when self-appointing governing groups grow insular and unrepresentative of the entire project community (as member-selection processes tend to spawn self-reinforcing leadership cultures). Moreover, this model can stymie community participation in leadership activities, as community members often feel like they must "wait to be chosen" before they can take initiative on work that interests them.

To get started in a project with this governance model: Because this governance model is typical of more mature open source projects, communities adopting this model will often curate getting started documentation aimed at assisting potential contributors. Find this documentation and read it first. Then read the project’s governance documentation to determine how its governing bodies are composed. In many cases, you can locate a council or board governing the part of the project where you would like to make a contribution. That body will be able to oversee your contribution and answer questions you may have.

6.8.4. Electoral

Some open source projects choose to conduct governance through elections. They may hold elections for various roles, or conduct similar electoral processes to ratify or update project policies and procedures. Under the electoral model, communities establish and document electoral procedures to which they all agree, then enact those procedures as a regular matter of decision-making.

This model is more common in larger open source projects where multiple qualified and interested contributors offer to play the same role. Elections are also common for projects with a sponsor (a foundation, for example), because an electoral process can make the allocation of sponsor resources more transparent. Electoral governance also tends to lead to precise documentation of project roles, procedures, and participation guidelines. When election documents make these matters explicit, they help new contributors maximize their involvement in a project.

But elections also have drawbacks. They can become contentious, distracting, and time-consuming for all project members (whether those members are running or not). Some communities promote elections as a solution to the indefinite tenure of well-known project members; however, elections don’t generally cause turnover unless term limits are part of the project’s policies.

To get started in a project with this governance model: Communities appointing leaders through elections typically feature election results and a leadership roster prominently on their project websites. Review those documents to determine a point of contact in the project. Well-governed open source communities will it make clear on their project websites their processes for proposing and reviewing items that the community can vote on. As you establish a reputation for making useful contributions to the project, you may eventually decide to be a candidate for a project leadership position. Be sure to interact productively and collaborate effectively with other contributors as they may be voting you into a leadership position some day.

6.8.5. Single-vendor

Occasionally, individual companies or industry consortia may choose to distribute software under the terms of an open source license as a way of reaching potential developers and users—even if they do not accept project contributions from those audiences. They might do this to accelerate adoption of their work, spur development activity atop a software platform, support a plugin ecosystem, or avoid the overhead required for cultivating an external developer community.

Under this model, the governing organization usually does not accept contributions from anyone outside it. Instead, open and closed source innovation occurs at the edges of the project, just where it contacts the rest of the world. For this reason, some commentators call this the "walled garden" governance model. Occasionally, projects following this model will adopt license with strong "copyleft" requirements, which they see as a deterrent to commercial competitors benefitting from their work on the project (the goal is to force competitors and customers with production requirements to purchase a non-open source license for the software—what some call a "dual-license" approach). This model becomes problematic in cases where a project claims to have an open community but is in fact wholly owned by a company or consortium.

To get started in a project with this governance model: First, consider any existing relationship between your employer and the company originating the project, if applicable. Next, assess the project's licensing terms and review its change history and bug tracker to determine whether you are able to contribute to the aspect of the project that interests you—and in the way you would like. Given the project's particular licensing stipulations, you may find yourself working alongside or on top of a particular project rather than contributing to it directly.

6.8.6. Foundation-backed

To exert greater control over resources and project code, some open source projects choose to be managed by an incorporated NGO (non-government organization), such as a charitable nonprofit or trade association. Doing this allows the "project," as an abstract entity, to take ownership of resources like servers, trademarks, patents, and insurance policies.

In some cases, foundation leadership and project leadership can form a single governance structure that manages all aspects of the open source project. In other cases, the foundation manages some matters—such as trademarks and events—and other governance structures in the project(s) control other matters (such as code approval).

Extensive funding and legal requirements normally limit this model to larger open source projects. However, many smaller projects choose to join larger so-called umbrella foundations, such as the Software Freedom Conservancy or the Linux Foundation, to reap some of the benefits of this governance model. This governance model is advantageous for projects seeking to establish legal relationships with third parties (like conference venues) or projects seeking to ensure successful leadership transitions following departure of key individuals. It might also help prevent the commercialization of the project under a single vendor.

High overhead—not strictly financial, but particularly in terms of contributor time, which can be substantial—is a significant drawback of the foundation-backed governance model. Some foundations are incorporated as industry consortia, in which sponsoring companies govern the organization. Different consortia allow different degrees of participation from individual project contributors; some are fairly open groups, while in others only corporate managers have authority.

To get started in a project with this governance model: If a foundation does not govern day-to-day project contribution activity, then locate the project's getting started documentation and follow it. Otherwise, note that individual projects under a particular foundation's umbrella will have their own sets of leaders, though some common guidelines may standardize basic contribution processes across all projects a foundation governs. To identify a specific project's leaders, consider addressing a request to the foundation members' mailing list. You might also examine the project's change history to identify frequent contributors and contact them. As many foundations feature a contribution-based voting system, familiarize yourself with steps required to become a full voting member of the foundation. If the foundation is a members-only industry consortium, determine whether your employer is already a member. If not, talk to your manager about the importance of the project to your work and ask whether your employer might consider joining. In either case, foundation projects may require signing contributor paperwork. Your legal department should assist with reviewing and signing such paperwork.

6.9. Conducting basic governance

So far, we've discussed the nature and importance of open source project and community governance, factors that trigger evolutions in project governance models, and a few of the most popular open source governance models. Finally, let's examine some concrete steps you can take to structure your own community's governance—whether you're launching a new project or evolving one that's already active.

Recall that most governance models consist of two primary dimensions: roles, and policies and procedures. The basic requirements here are actually quite spartan, and can be evolved as the project grows. What follows constitutes a kind of *minimum viable product* for project governance.

In your project, each of the following sections could very well be its own document. Or they might simply be part of a single long README—or anything in between. What's important it to get the basics of how things work down in text, so that people thinking about participating in your project know where to go, who to talk to, and most of all aren't horribly surprised.

6.9.1. The importance of honesty

When writing governance documentation, it can be tempting to define your project as you would like it to be—or how your corporate marketing department would like it to be seen—rather than how it actually is. Particularly, project leaders frequently make the mistake of attempting to make the project appear more democratic than it actually is, in documentation. This falls apart when users or contributors expect your project to live up to its governance documentation, and it doesn't. People who would have been fine with being told a project was single-company at the outset become very upset if they ask for their committer status and are refused later.

Like technical documentation, governance documentation should explain how things actually work. If there are aspirational goals, those go in their own section under "Roadmap" or "TODO."

6.9.2. Defining roles

As mentioned, your project will have a variety of real roles, but you only need to define a handful of them to start out. Those basic Roles are:

1. *Member*
2. *Contributor*
3. *Leader*

Whether you've thought about it, your project already features all these roles you already have in your project. Each one of them should be recorded in a roles document of some kind, either in your project's documentation or your main source code repository. This allows you to make what was implicit into explicit, both setting expectations for and allowing more people to participate in your project. For each role, you'll need to define who they are, how they qualify for that role, what they are expected to do, and what their rights and privileges are. Eventually you'll go beyond these roles and define many more specific ones. But detailing these three will take your project a fair distance on its journey.

Members

This is possibly the least-documented role across all of open source, despite being the most pervasive. Members are the people or organizations who participate in your project and are recognized for it. Depending on how your project is run, these can be subscribers on a mailing list, sponsoring companies, known end-users, participants at an event, or members of a foundation. In some projects, Member is synonymous with Contributor, but in most this is not the case. Most projects have a much larger cadre of people who are involved with the project in some way but are not actively contributing code or content to it.

Defining who Members are requires deciding who the project is actually serving, which is always a critical discussion to have. Are customers of the main sponsoring company automatically project Members? Can companies be Members, or only individuals? Are end-users Members or can they only be Contributors? More than anything, defining Members means defining who it is that project Leaders need to listen to.

For almost all projects, you need to specify what rules Members are subject to (usually a code of conduct and not much else) and what they can expect from Leaders and Contributors. It's particularly helpful to explain how Members should participate in the project, such as "Members file bugs against this repository, and use the 'new bug' template." Most people, given clear instructions, are happy to channel their participation into the routes you show them.

In projects with democratically elected leadership, Members can be a much more rigorously defined role, because being a Member can come with voting rights. This requires you to more carefully qualify Members to avoid vote-packing or simply derailing election procedures.

Contributors

Far more projects have a written definition of Contributors, but fewer than you'd think. It's often assumed, in the age of publicly hosted source code control, that you count anyone in the GitHub or GitLab statistics as therefore a Contributor. But defining "who is a Contributor to this project" can be deceptively hard.

Is it anyone who posted on a mailing list, or do you need 100 merged pull requests? Is it just code contributors, or contributors of any kind? What about folks who do events and advocacy? Are staff who work for a contributing company automatically considered Contributors, or do they have to earn it individually? What about someone who contributed a lot of code three years ago, but not since then? Who gets listed in your release credits and how?

The conversation around this will often have a greater effect on your project than the document does.

The Contributor role is also one for which you'll need to set many more expectations for what Contributors receive in return for their work. This not only includes an explanation of the intellectual property rules of the project (e.g., does the contributor still own their code or not), but also questions like how soon Contributor can expect their submissions to be reviewed and accepted or rejected. Generally, you should also explain how the Contributor will be credited for their participation.

It's also a place where you set out clearly what rules Contributors need to follow. For example, some projects require Contributors or their employers to sign paperwork officially sharing their copyright or other intellectual property (see below for more on this). You may also require Contributors to do certain things to help maintain the project, such as review others' submissions or help with documentation.

Leaders

As we noted, every project has leadership, even when those leaders are not clearly identified. As such, at a minimum you'll need to transparently identify who your Leaders are, so that decision-making processes can be clear. Many projects also explain the qualifications and procedure to become a Leader, whether it's selection by a committee, election, or simply based on your job. If you have a more politically sophisticated project, then those should be written down in a selection/election procedure document as well (refer below), but if it's simple, selection can just be part of the role document.

What fewer projects put into their leadership role documents is the other parts: the powers and limitations of the Leaders, their duties, and how people leave the role (voluntary or not). It's very important that everyone know exactly how far a Leader's authority extends, as well as what they're responsible for, or you end up with a lot of conflict between Leaders and other project members. Having a set of written duties helps immensely when your leadership team has to decide to remove a project Leader who has stopped participating, but does not want to resign.

If your project is trying to recruit new/additional Leaders, then it's also important to have a detailed set of qualifications a Leader needs to meet. Contrary to some expectations, having detailed qualifications gives people who want to move up in the project a target to shoot for.

6.10. Setting policies and procedures

In addition to some basic role documentation, there’s a certain amount of basic paperwork that each project should create for itself. These policy and procedure (P&P) documents are considered a kind of minimum for what you need in order to grow and mature a project. Your project may, and eventually will, have other P&P docs as your contributor base expands and the number of processes you need to write down with it.

Some of these will be mostly technical (like release process, or a support policy), and we won’t be exploring those here.

However, there are three governance P&P that every project should have:

1. Code of conduct
2. Contribution process
3. Communication information

Projects that grow larger and more popular, become commercially adopted, or are actively recruiting many new contributors probably want some additional P&P documents, such as:

1. Leadership selection/election process
2. Contributor promotion
3. Release process
4. Security issue reporting and handling
5. Project trademark usage

We’ll talk about these eight documents below.

6.10.1. Developing a code of conduct

Creating a code of conduct (CoC) for your open source community is one of the simplest and most powerful ways to begin influencing the project’s governance model. A code of conduct is a description of expectations for community members' behavior when they act within or on behalf of the project. It might outline the values a community agrees to uphold, articulate the behaviors community members expect one another to exhibit in the service of those values, and identify the consequences of violating the code. The most effective codes of conduct are those written through collaborative processes that involve participants across the community (not just project leadership!). In this way, constructing a code of conduct can become a compelling community-building exercise.

Here are the core items that every Code of Conduct needs to have:

1. A statement of what kind of behavior is encouraged
2. A statement of what kinds of behavior are prohibited
3. Contact information for reporting violations
4. A description of the enforcement mechanism

When you're starting out, both the report recipients and the enforcers of the CoC are likely to be your project founders. As your project grows, you'll want to form a specific CoC committee, but you don't need that right away.

6.10.2. Contribution process

In order to recruit contributors, you need to tell them the basics of how to contribute to your project. For projects on GitHub or GitLab this is generally placed in a document called CONTRIBUTING.md, but it can really go anywhere as long as it's linked from your project's home page. If you've documented your Contributor role, you can just use that for your contribution docs. If you haven't, then here's a few things you should cover in your contribution document:

1. Where to communicate with other contributors.
2. How to submit your first code, documentation, or other contribution.
3. Any testing or formatting requirements, in detail.
4. What to expect from the review process.
5. When they qualify for membership/contributor status.

Some projects have paperwork that needs to be submitted before any contributions can be accepted, such as a Developer Certificate or Origin (DCO) or Contributor License Agreement (CLA), certificate of identity, or GPG keyring. Spell these out with step-by-step instructions in your contribution document.

6.10.3. Communication information

Most open source projects have multiple ways that project members talk to each other, including email, chat, issues, code reviews, video conferencing, and even in-person meetings. You need to spell out which channels your project uses, and how to join them. It's also important to keep this information up to date.

If you have them, it's useful to list both your user forums as well as the channels used for contributors, so that people know where to take their questions. Distinguish the media used for official project business as opposed to unofficial channels used for general discussion. It's extremely frustrating for contributors to be told "oh, we decided that on the mailing list" if they didn't even know there was a mailing list. Any regular meetings should link to a calendar, or at least information about the next meeting. And if your community has any important events, such as annual developer conference, mention it.

Refer to this guidebook's chapter on communication norms in open source projects for more detail.

6.10.4. Leadership selection/election process

If you've already documented your "leadership" role, the information on how project members become leaders will be part of it. However, some projects don't get around to writing roles, and other projects have multi-step election procedures that require additional documentation. Some just want a quick-reference of how the election or selection process works.

If you have the typical new small project, this document will be very short indeed, containing simply the list of project leaders, who are also the project founders. If your project has a self-appointing council, it's not that much more complicated; just write down how the selection works.

Projects that have full-blown elections will need a longer document containing all of the provisions of elections, including who gets to vote, how the vote is conducted and by whom, what the schedule is, and how candidates are selected. We'll offer additional advice on holding elections at the conclusion of this chapter.

6.10.5. Contributor promotion

If your project has multiple levels of contributor status, with a defined progression between them—what's known as a "contributor ladder"—then it can be useful to write a specific document explaining how this works. This will give new contributors an idea of what's ahead of them and what they need to do to move up. It also helps make sure that contributor promotion is being done fairly.

For fairness, it's preferable to make the promotion rules as objective as possible. For example, "Has consistently helped with code reviews in the subproject" is good, but, "Has completed at least 40 code reviews over the last 3 months in the subproject" is better. Quantifiable rules help you avoid overlooking contributors who are valuable, but not outspoken.

Smaller projects, with only a couple of contributor levels (e.g., Contributor and Owner), do not need a separate document for this.

6.10.6. Release process

Releasing software involves making decisions around what will and won't be included in the current release. When a project is small, this is pretty obvious, but in larger projects with contributors working for multiple employers, deciding what stays and what gets cut can be political. Decisions about which platforms are supported can also be contentious. As such, when your project grows you're going to want to write down some process around releases.

Some projects have defined release teams, in which case this document will be largely a collection of Role documents for the release team. In other projects, the maintainers do the releases, but even with those it's worthwhile to explain how they decide what gets included. This doesn't mean necessarily changing how you do releases, but rather just writing down what the real procedure already is, particularly the method of deciding which features and patches get left out. The process for writing and editing the release announcement is also worthwhile, especially if your project involves multiple vendors.

This document will also have lots of non-governance content, like the locations of the servers, the commands to build packages, and how long to wait for mirrors to sync. It's expected that most of it will be technical instructions. Just don't neglect the *who* and *why* along with the *how*.

6.10.7. Security issue reporting and handling

Once your project's code is being used in production by external users, managing security issue reports becomes a critical priority. While this topic could use an entire chapter on its own, there is some basic governance setup associated with handling security issues. This will include:

1. Who is selected to be on the security team, how, and when.
2. Where security reports get sent.
3. How they are handled, including confidentiality requirements.
4. What reciprocation security researchers can expect.
5. How long you can wait before disclosing.

Confidentiality requirements are particularly important for both the security team itself, and for the programmers and security researchers with whom you work. For example, security researchers are willing to not disclose their findings to the public until your project does, but only if they are promised that your security team won't do that either. In many projects, security team members aren't allowed to share certain information even with their own employers.

6.10.8. Trademark usage

When a project gets popular, both commercial and non-commercial groups want to use the project's name, word mark, and graphical logo. Whether it's just statements of support, a third party wanting to sell shirts with your project on them, or other projects that derive from yours, projects need these entities to be following some kind of official policy around usage. Even if the project has not filed for a trademark with any government yet, establishing a pattern of policy and permission will help protect your project's name and marks in the future.

Such a policy consists of four things:

1. A general statement of acceptable usage.
2. Contact information to request specific permission or for clarification.
3. A designated team, committee, or contributor who is going to handle these requests.
4. Additional guidelines for the trademark team.

For the actual acceptable usage statement and guidelines, projects should obtain legal assistance. The governance part of this is selecting the "trademark team" (which could be an existing steering council, or similar), and how guidelines are updated and changed. In projects run by multiple technology vendors, it's critical to work this out in the early stages of the project, because the project's own sponsors will want to use its mark almost immediately. Make sure that responsibility here is shared between the stakeholders in your project.

Like security issues, the trademark team needs to be able to handle confidential contacts, because sometimes pre-release startups may want to use your project name.

6.11. Holding community elections

As community projects grow, many choose to select community representatives. This process may occur when a community loses a founder, a group decides to move to a "ruling technical council" governance model, or when a project moves to a non-profit governing body with paying members.

Regardless of the circumstances, many projects opt to select their community representatives through elections. Historically, choosing a voting system, defining an electorate, and limiting the pool of eligible candidates has proven complicated for community projects.

This section summarizes project election best practices, including who gets to vote, who can be a candidate, and how elections are run.

6.11.1. Electorate and eligible candidate pool

Establishing franchise rules is critical. Some projects have allowed anyone registered for a project's site to vote in an election—a very low bar—but have specified that only project committers could be election candidates—a high bar. However, almost all projects eventually broaden the pool of potential candidates to equal the pool of voters. Anyone who can vote in the election is therefore eligible to become a candidate.

Projects typically take one of three approaches to defining the electorate and candidate pools:

1. **High bar:** Voters are members of an inside group—such as committers, maintainers, and core contributors. Membership requires a long history of participation and seniority recognized by peers.
2. **Medium bar:** Active participants or foundation members can vote, as long as they meet a clearly articulated definition of participation.
3. **Low bar:** Anyone can vote as long as they complete some basic steps, like signing up to the program or joining the mailing list.

Defining an activity metric and minimum bar specifying what qualifies as "participation" can become contentious, mainly because it involves drawing arbitrary lines delimiting eligible participants. Generally, projects specify that quantified, ongoing participation is necessary to become part of the electorate.

One common election fear is ballot stuffing or cohort effects, where large companies dominate the representative bodies by having a large voting bloc, or where friends of candidates will pass the low bar to become voters simply to vote for their candidate. In most cases, however, such fears are unfounded. Technical communities often try to create rules to mitigate against possible abuses of the system, but in most cases, these rules are "premature optimization," which Donald Knuth, author of *The Art of Computer Programming*, has famously described as "the root of all evil." [9] Avoiding special rules—and addressing issues with the electoral process as they arise—is generally the better practice.

One final consideration is the process for becoming a candidate in the election. The most popular option is self-nomination, where candidates post election information and their reasons for running. Another option is nomination, which is often the same as self-nomination as the candidate typically asks people to nominate them and second their nomination.

6.11.2. Voting system

Another complex community decision is the voting system. Any community will include people passionate about how to vote—and how to count votes. Without proper care, conversations about these issues can go on for months and result in proposals that are almost impossible to implement.

Most community projects have used:

1. Voting by secret ballot.
2. Online voting, with a personal token to ensure each person may only vote once.
3. Some form of preferential voting, listing candidates in order of preference.
4. Condorcet or single transferable vote (STV) to count the votes and identify winners.

Some projects continue to use alternative voting systems like "first past the post" or weighted voting systems, in which voters receive 12 tokens to allocate to candidates however they wish, and the candidates with the most tokens win the election.

Several projects use online counting software. Options to consider include:

1. Condorcet Internet Voting Service, a free, online voting and Condorcet counting system.
2. OpaVote (formerly OpenSTV), a commercial election counting Software-as-a-Service.
3. OpenSTV, formerly available under the General Public License (GPL) and still used by several projects to count elections.
4. Helios, another free election service that allows online voting and several different vote counting methods.

6.11.3. How to start

If you are planning to propose an election system, begin with a mission statement. For example:

> The goal is to ensure the technical steering committee represents everyone contributing actively to the project, valuing non-code contributions equally to code contributions, in the definition of the technical scope and direction of the project.

The mission statement clarifies several things: who is being represented by the elected body, what their authority will be, and why they are being elected. Once you have agreed on the goal of the elected body, choose the simplest ways to define membership in the body being represented. Then, choose the simplest voting and counting system possible.

[9] Knuth, Donald E, *The Art of Computer Programming*. Reading, Mass: Addison-Wesley Pub. Co, 1968. Print.

Chapter 7. Understanding Community Metrics

7.1. Introduction: What’s a "healthy" open source community?

A healthy open source community is one that demonstrates open practices, uses open infrastructure, and cultivates an open culture with the goal of becoming more sustainable. But even for the most seasoned community managers and community architects, measuring an open source community’s health is a complex, difficult, and occasionally intimidating task.

That’s because any picture of project health is actually a mosaic; multiple factors combine to depict a community’s overall health. You’ll never find just one indicator you can point to and say, "See? The community is healthy."

What’s more, the factors that determine overall project and community health are always changing. In the past, for example, some may have pointed to "total number of download" as a health metric. But today we understand that even millions of downloads does not reflect the fact that a communuity’s production and quality assurance processes might be out of sync—or that *trolls* are misbehaving on the project’s mailing lists. A more complete picture would consider not only *results* or *outcomes* (like downloads, release cadence, etc.), but also *processes* (onboarding barriers, how easily a project can discover and promote maintainers, etc.).

This chapter explains in more detail the importance of a metric-driven approach to building, maintaining, and growing an open source community. It outlines a few important factors to consider when assessing an open source project’s and community’s overall health. And while every commuity is different—and therefore requires different metrics for measuring success—this chapter offers a few example metrics you might consider when measuring your own community’s successes.

7.2. Why community metrics are important

Establishing and maintaining community metrics is important for several reasons, including:

- Providing objective measures
- Promoting a culture of transparency
- Encouraging community participation
- Identifying potential community bottlenecks

Let’s explore each of these in more detail.

7.2.1. Providing objective measures

For any organization, you want to have a set of objective measurement to help you understand how you are performing. For businesses, it could be things like revenue, new customer aquisitions, costs, etc. that you can look at periodically (e.g., quarter-over-quarter) to see how you are progressing. For open source projects, it is also helpful to have a set of metrics that can help measure the health of your community. These could include things like number of contributors, how long it takes to close bugs/issues, responsiveness of reviewers, contributor diversity, etc.

7.2.2. Promoting a culture of transparency

Transparency is a key ingredient in all open source projects, so it is important to have insight into all contributors to your project. This isn't necessarily to rank or highlight the most active contributors. Rather, this is to get a better sense of where the contributions are coming from (e.g., different geography, organizations, etc.). When you start going beyond 20-30 community members, it starts to become more challenging to have a good insight into all contributors if you don't have a good set of metrics.

7.2.3. Encouraging community participation

Community metrics will not necessarily provide a full picture of the community, but they do offer important insight into the state of the community. For example, metrics can show if there are regular activities on project's code, bugs/issues, documentation, etc. for the project. Also, when people join a new community, they typically struggle with identifying and connecting with more experienced members. If the metrics dashboard includes a list of active documentation contributors, that could be useful for anyone interested in contributing to project documentation. Some communities even have a dashboard of *first time* contributors to highlight new community members. These may seem like small things, but they could go a long way to help new community members onboard into a project and feel welcome.

7.2.4. Identifying potential community bottlenecks

As open source projects grow, you will typically face challenges with responsiveness to community members. For example, code reviews may take longer or you may see a growing backlog of issues. If you have a good set of metrics (e.g., average time to close issues, median time for code review, etc.) that you review on a regular basis, they could serve as early warning signs before you start to have a number of dissatisfied community members.

7.3. Basic considerations

Before you determine how you will measure project and community health, you'll need to determine what you're going to measure. You might begin by looking at:

- Project life cycle
- Intended audience

- Governance
- Project leaders
- Release manager and process
- Goals and roadmap
- Onboarding processes
- Internal communication
- Outreach
- Awareness
- Ecosystem

Let’s examine each one of these in more detail.

7.3.1. Project life cycle

An open source project’s life cycle affects many other considerations about a project’s health. Understanding the project’s place in that life cycle will help you contextualize your assessment (e.g., single-person governance can be common when a project is young—but less effective when a project is more mature). Monitoring contributor trends might reveal critical information about a project’s short-term or long-term future.

When assessing the project’s life cycle, consider questions like:

- When was the project founded, and how old is it?
- How often do new contributors join the project?
- How frequently does the project accept new contributions?

7.3.2. Intended audience

Well-run open source projects demonstrate a clear understanding of the users (and contributors) they hope to assist and engage. When assessing a project’s intended audience, consider questions like:

- Has the project clearly identified a intended audience, and who is it?
- Is the intended audience the most appropriate one for this project?
- Does the intended audience engage with competing or complementary projects?

7.3.3. Governance

Governance refers to the rules and customs that define who does what in an open source project and how they are supposed to do it. Healthy projects entail thoroughly documented (and continuously evolving) governance models. When assessing a project’s governance, consider questions like:

- What is the project’s governance model, and is it publicly documented?

- Does the model account for both technical and business concerns?
- How do project members make and enforce decisions?

Refer to this guidebook’s chapter on governance for more consideration of this topic.

7.3.4. Project leaders

In healthy projects, leaders are visible and easily identifiable. Leaders often coordinate project work and establish a project’s vision, and usually have extensive knowledge of project history. When assessing a project’s leadership, consider questions like:

- Who are the project leaders?
- What are the project leaders' responsibilities, and are they focused more on engineering, marketing, or some combination of both?

7.3.5. Release manager and process

In healthy projects, members have formally documented release processes and identified release managers to supervise those processes. When assessing a project’s release manager and process, consider questions like:

- Is the project’s release process documented?
- Does the project have an identified release manager?
- How often do project release updates occur?
- Do project releases occur on a steady and predictable schedule?

7.3.6. Goals and roadmap

Healthy open source projects have publicly shared goals and clear processes for reaching those goals. Goals are attainable and clear deadlines exist for tracking progress toward those goals. When assessing a project’s goals and roadmap, consider questions like:

- Are project goals clear and public?
- Does the project have a clearly communicated process, and is it also public?
- Do project participants have a history of meeting project deadlines?

7.3.7. Onboarding processes

New contributors are vital to project innovation and success. Healthy projects feature clear, welcoming onboarding materials that assist newcomers who wish to participate in the project. When assessing a project’s onboarding processes, consider questions like:

- Does documentation explain precisely what the project is and how to use it?
- Does documentation help new contributors get involved in the project?
- Does the project accept contributions of more than one type (e.g., development, marketing, project management, event planning)?

7.3.8. Internal communication

Communication channels are key indicators of project health, as are a project's internal communication practices. Issues affecting community health often emerge first in internal channels—such as mailing lists or chat platforms—where contributors and users interact. When assessing a project's internal communication, consider questions like:

- Does the project have sufficient communication channels?
- Can people find and use these channels effectively?
- Are channels regularly moderated?
- Is channel communication governed by a code of conduct?

Refer to this guidebook's chapter on communications norms for more consideration of this topic.

7.3.9. Outreach

Outreach is the process of actively promoting a project and making others aware of it. Communities use written materials (e.g., social media, blogs, whitepapers), events (e.g., meetups, conventions), and educational tactics (e.g., demos, training sessions) for outreach. Healthy projects have adequate energy and resources devoted to outreach. When assessing a project's outreach efforts, consider questions like:

- Does the community use clear and consistent methods for outreach? If not, does it plan to establish a set of outreach methods?
- Are people writing, talking about, and promoting this project and its technologies?

7.3.10. Awareness

The project's intended audience must be aware of the project and understand the problems it solves. Awareness is a desired outcome of a project's outreach efforts and can be measured through user and contributor surveys or general marketing analyses. When assessing a project's awareness, consider questions like:

- Is the intended audience aware of the project?
- Can people in the intended audience explain the project's uses, features, and advantages over alternatives?
- Do others working in an industry that would benefit from the project know the project exists?

7.3.11. Ecosystem

No project exists in a vacuum. Projects frequently depend on one another. In some cases, similar projects can be competing to reach the same intended audiences. A community's interactions with other projects in its ecosystem reflect the project's health. When assessing a project's ecosystem, consider questions like:

- What are the project's dependencies and what projects depend on it?
- Is the community sufficiently integrated into the overall project ecosystem, intended industry, and organizations that may use the project?
- Do members of that ecosystem view this project favorably?

7.4. Choosing the right metrics for your community

Because no two open source communities are the same, every community will naturally have its own set of metrics for measuring health and success. Many factors can influence a community's choice of metrics, but one of the most important influences is the community's goals.

Some communities, for example, prioritize how quickly they're able to merge code—so they track metrics related to this ability. But other communities consist of users and contributors working in heavily regulated industries (like energy or health care), where necessity dictates that decisions to merge new fixes and features take a longer time. Those communities probably wouldn't emphazize speed of code review as much as, for example, code review efficiency.

Depending on its goals, a community might establish metrics for measuring things like work backlog, contributor diversity (e.g., organizational, geographical, etc.), flow of first-time contributors, localization and internationalization, or popularity of discussion topics. Communities should always establish metrics collaboratively and agree on them collectively. Hearing from a diverse group of community members is important for ensuring the metrics are inclusive and not just focusing on the work of a subset of the community.

It's also a good practice to review the metrics periodically with the community and discuss if any adjustments are needed for your metrics. Even within the same community, you will likely need to evolve your metrics along with the community as your needs change.

7.5. Resources for developing metrics for your community

7.5.1. Take advantage of available resources in your software tools

In the past, many people wrote complex scripts/queries to get metrics for their communities. Nowadays, most of the software tools (code repositories, forums, issue trackers, wiki's, etc.) that open source projects typically use have APIs, plug-ins, or even built-in dashboards that makes it easy to collect data for your community. So if you use tools like Discourse, GitHub, GitLab, Jira, or others, you may be able to save a lot of time by reading their documentations prior to implementing a new set of metrics from scratch.

7.5.2. The CHAOSS project

Not surprisingly, there's an open source project that is focused on community metrics. The project is called CHAOSS (Community Health Analytics for Open Source software) and it has community members from academia, companies that participate in many open source projects, open source foundations, and others. If you visit the CHAOSS website, you will find details on metrics across different categories plus implementation examples for many of these metrics.

If you browse through CHAOSS metrics, you will likely find plenty of metrics (and implementations) that will be applicable to your community. If you have an idea for new metrics that are not yet in CHAOSS, you can also start a discussion on new metrics in the CHAOSS community.

7.5.3. Resources/examples from other communities

A number of open source communities have good documentation/code for their community dashboards that many other communities can take advantage of. Many readers may be familiar with the CNCF dashboard and you can find details on their `devstats` project for their dashboard in the CNCF's repo at https://github.com/cncf/devstats.

Another good example is the Ruby community dashboard and their FAQ page, which provides good insight into why they developed the dashboard and some of their implementation decisions.

Some communities publish contribution metrics after each release. Here is a good example from WordPress after their recent release, where you will see a lot of good visualizations for where the contrbutions are coming from.

7.6. Metrics pitfalls

Metrics are certainly important, but there are definitely shortcomings that we need to be aware of.

- People often measure the most easily measurable in their metrics: this is human nature as we all want to do what's easier. However, you run the risk of neglecting important aspects of the community if you only focus on easily mesurable metrics in your community. For example, it's often easier to focus on inputs (e.g., number of commits/merge requests/pull requests) compared to outputs (e.g., the impact of a commit/merge request/pull request). Needless to say, ignoring outputs from the community will provide an incomplete picture of the community.
- Over-reliance on metrics will provide an incomplete insight: No set of metrics will provide a full picture of communities (or any organizations for that matter). Although it is important to have a standardized set of metrics so that you can gauge your community's progress over time, there will always be things that are extremely difficult to measure or quantify. For example, we all want contributors to feel a strong sense of belonging in the community and enjoy collaborating with other community members. Whether this is happening or not would be difficult to quantify,

but you still want to have a good sense on this aspect of community health even if it requires other means besides metrics collections.

- Ignoring intrinsic motivation: people often join (especially volunteer) organizations because they are intrincically motivated. For example, they strongly identify with group’s mission or enjoy a sense of belonging with other members. If there is too much emphasis on people’s contribution (or input) in metrics, you run the risk of losing sight of why people joined the organization in the first place. Most contributors in open source communities are volunteers who contribute in their own time, so ignoring their intrinsic motivation can often lead to negative consequences in the community.

Refer to the chapter on participant motivations for more on the topic of intrinsic motivations.

Beyond these shortcomings of metrics in general, the following are particularly relevant to open source communities.

- Gaming contribution metrics: this usually happens when metrics are used as a main (or even a sole) basis for recognition in the community. Not suprisingly you will see behaviors like people submitting multiple commits/merge requests/pull requests for trvial changes when they could have accomplished the same thing with a single commit/merge request/pull request.
- Vanity metrics: you also see vanity metrics outside of open source. A good example is placing too much emphasis on things like the number of social media followers. As in social media, quantity isn’t everything. Also, if you want to ensure that community members' intrinsic motivation is satisfied in the community, vanity metrics is definitely not a good way to go.
- Making comparisons between different open source communities based on a few metrics: sometimes you will hear things like, "Project A had more than 5,000 attendees in their last conference," or, "Project B has 1,000 contributors," and people have similar intentions for their community. Before you are tempted to compare your community to others, it’s important to consider if you are making apples-to-apples, that is, a like-for-like comparison. You may be in a different industry, in different stages of project maturity, or have a different scope, etc. and a direct comparison may not be appropriate. Before you think about wanting 1,000 contributors in your community, you may want to ask basic questions like do you really need 1,000 people to accomplish your project’s goals?
- Too much focus on code contributions: it may be because there are more tools available to capture code activities, but there’s a tendency to focus mostly on code contributions in open source communities. However, it is important to remember other valuable contributions such as answering questions on forums, triaging issues, maintaining wiki pages, etc. There should be an effort to ensure that community metrics reflect a variety of contributions (both code and non-code) so that no one in the community feels left out.

7.7. Metrics dos and don’ts

Finally, here are some dos and don’ts when you are working with open source community metrics.

7.7.1. Dos:

- Do make metrics public: This may be stating the obvious, but transparency in open source should also extend to metrics. When you develop metrics, it helps to include a diverse group of people in the process so that metrics are inclusive and consider all contributions. Also, if any adjustments need to be made for your community metrics, it’s likely that we will first get that feedback/suggestion from community members. Also, all metrics should be open to everyone so there is confidence in the data.
- Do use metrics for spotting outliers: metrics are particularly useful for highlighting areas that aren’t doing well. Good examples are things related to throughput such as time it takes for close issues, forum posts to be answered, code review, etc. In these examples, metrics are a great tool that can help identify potential bottlenecks early.
- Do use metrics as a starting point for gaining further insight into community health: Metrics may tell you *what* is happening in your community, but you typically will not know the *why* just by looking at the numbers. If the metrics shows that the number of first time contributors are declining, you will probably need to have some one-to-one hallway or phone conversations in order to identify the causes of the decline. Metrics will highlight the symptoms as a starting point, but people will then have to do the work from there.

7.7.2. Dont’s:

- Don’t use metrics as a sole basis for rewards: we already discussed gaming of contribution metrics previously if you only rely on metrics for rewards in the community. In addition, if people perceive that rewards and recognition are mostly based on the volume of work (or input), you run the risk of discouraging people who aren’t able to devote as much time to the project or people who are getting started in the community. People do not join open source communities just to do more work and we do not want to lose sight of their intrinsic motivation.
- Don’t present metrics without a proper context: even when you get asked what sounds like a straightforward question like "how many contributors do you have in your community?", it is always helpful to get some context behind the question. Depending on who is asking the question, they’re usually asking for something slightly different. For example, the total number of contributors in project’s history may be appropriate in one context, but in another the growth of contributors over a time period may be what the questioner is really after. Also what do they mean by contributors? Do they also want to include people contributing to internationalization, issue triage, event planning, etc.? So before we simply point people to a set of metrics, it is helpful to understand the context or even motivation behind their question.

- Ignoring non-metrics: As discussed previously, not everything is easily measurable or quantifiable. Even if we have a well defined and polished metrics dashboard, it should not stop us from continuing to have human conversation with community members to keep a pulse on what is happening in the community and encourage community members to point us to what we are not able to see in our metrics.

Chapter 8. Community Manager Self-Care

8.1. Disclaimer

Materials in this chapter are for informational purposes only, not for the purpose of providing medical advice. This chapter is not intended as a substitute for professional medical advice, diagnosis, or treatment. Always seek the advice of your physician or other qualified health care provider with any questions or concerns you may have regarding your health, both mental and physical.

8.2. Introduction

Because of their roles (but also, perhaps, because of their natures), community managers often focus (even fixate) on the health and well-being of their community members and teams. They understand the challenges and difficulties associated with encouraging stable, safe, and inclusive online communities, places where people demonstrate compassion and empathy. But too often, they mistakenly assume they're immune to the same emotionally difficult challenges they strive to help other community members overcome.

What's more, they may even try to *hide* their vulnerabilities, for fear of being perceived as less capable of performing their roles—or, worse, being rejected by their communities or peer groups completely. This perspective on vulnerability can lead community managers down an isolating road, one that might make them feel as though they aren't able to reach out to seek help when they need it.

And yet, the longer we don't attend to our well being, prolonging treatment for our increasingly negative symptoms, the larger the problem—until, eventually, we are mentally and physically debilitated, suffocated, and denied of the fulfillment and joy our role once provided us. Regaining that joy could take months, even years. In fact, we may never regain that same level of passion for the role again.

To help ourselves as community managers, and in turn help others, we need to learn how to spot the signs of dissatistfaction, disillusionment, and burnout. These signs can be subtle and hard to detect, but by identifying and acknowledging these changes in behavior (first and foremost in yourself), you begin applying similar awareness to others in your communities. This experience can provide insight to developing and building up social norms and mechanisms for the community to help reduce the occurrence and impact of burnout, thus promoting a physically and mentally healthy and more productive community.

In this chapter, we'll explore the importance of self-care and why community managers should be mindful of the interpersonal aspects of the role that may affect them both positively and negatively. This chapter also provides practical steps to encourage and maintain a healthy work life balance that can be applied to themselves and promoted within their community.

8.3. The importance of self-care

What exactly is self-care?

The WHO (World Health Organization) defines self-care as:

> "Self-care is the ability of individuals, families and communities to promote health, prevent disease, maintain health, and to cope with illness and disability with or without the support of a healthcare provider".[10]

Thus, self-care is the practice of making conscious, mindful efforts to perform activities that aid in our physical, emotional, relational, and perhaps spiritual well-being at a fundamental level. Most of us already practice self-care daily by taking actions to protect our overall health (for example, eating healthy, exercising, and getting enough sleep). Sometimes, however, we may find acts like these alone are insufficient for maintaining our well-being—and that a conscious effort at finding additional support is required.

We can usually handle stress more effectively when we're in a happy state of mind. By practicing self-care, we can build a resilience to life's unavoidable stressors.

This concept of self-care may appear simplistic in theory, but it is commonly overlooked or neglected—especially by community managers. Let's explore (and deconstruct) some reasons why.

8.3.1. Emotional contagion

Community managers are often seen as leaders, the people everyone can turn to, an inspiration for others to emulate. This seemingly high pedestal can be daunting, demanding a good deal of emotional intelligence and mental effort to deliver a community that's not only growing but also productive.[11]

When the going gets tough, it's often our responsibility (and our desire) to pull the community through those times. But in order to do this, we need to be in top condition *ourselves*. That is, we first need to look after ourselves before we can look after others in any sustainable way. Otherwise, we'll never be able to adequately face whatever is thrown at us—and something always is, when interacting with diverse personalities, resolving conflicts, making tough decisions, all while working towards future long term objectives. And that is why self-care is so important.

One of the first tasks on the road to self-care, then, is to be aware of how our own emotional states can impact our communities.

Emotions (both positive and negative) are highly contagious.[12] Our emotional expressions have the power to unconsciously influence people around us, such that others may even begin unconsciously mimicking our own.[13] If we are in a state of anxiety or stress, others will sense this and add to it with their own negative energy. We see this when dealing with particularly toxic community members infecting others, but fail to identify that we ourselves can unconsciously unduly influence others as well. Similarly, if we are in a genuine pleasant mood, then people tend to be driven by that positive mood and become more productive.

In short: In order to bring out the best in others we must bring our best to the table.

As leaders, we wield potentially immense power to make positive changes in our communities. Let's use this power. Take every opportunity to utilize your influence for encouraging and promoting self-care—not only for the community's benefit, but also for yourself. Educate yourself on what self-care looks like to you, and communicate that understanding with your team. Community managers don't need to share their most intimate fears or anxieties. Yet simply initiating a conversation on the importance of self-care can be beneficial, not just for the team *running* the community, but for the *entire* community.

Identify and implement regular, dedicated time in your and the team's workload to practice self-care. Set realistic expectations on deliverables and communicate frequently within the team the potential triggers of stress or strain that may require more downtime to focus on one's physical and mental health.

By taking steps towards actively promoting and advocating for self-care, you prompt members following your lead to do the same. You also begin revealing your human side, eroding the unhealthy assumption that leaders are invulnerable to emotional dimensions of the work.

8.3.2. Being (imperfectly and empathetically) human

We can often find ourselves being controlled by unrealistic generalizations (either our own, or others') of what a community manager should be: driven, friendly, successful, super-human, even perfect.

However, these impressions of what a leader really "should" be are full of contradictions and fundamentally flawed. A leader can be many things to many people, but the most important aspect is that they are human: flaws and all.

And yet community managers may tend to over-compensate, attempting to live up to these leadership images—especially by suppressing and hiding negative emotions for fear of being seen as "too emotional" or, worse, "out of control."

But there's a difference between being *in* control, being *controlled*, and being *under* control.

The belief that "being in control is a reflection of our confidence in what we produce" is inaccurate. It actually demonstrates that we are reluctant to show others that we are not perfect and struggle to accept anything less.[14] Although maintaining this front can help shield us temporarily from others' emotions and criticisms towards us, it doesn't protect us from our own self-destruction caused by bottling them up.

The consequences of suppressing your emotions will inevitably surface, and dramatically so. This is when our emotions *control us* (and our actions), which often leads to rumination, collapse of relationships with one's team or community, and other negative outcomes like burnout.

The ultimate aim is to be "under control," both with your emotions and your workload. Here, our passion and attention are directed in a constructive and purposeful manner—a particularly important place to be in when we've made a mistake.

The phrase "bring your whole self to work" is a good expression of allowing yourself this forgiveness. We should feel like we can admit when we don't know something, haven't made the correct decision, or have made a mistake. And we should also recognize and accept that we can have good days and bad ones.

Being honest with ourselves is as important as being honest with our communities. If we're not honest with ourselves and in tune with our emotional states, we may unintentionally escalate difficult situations within our communities (due to our failures at acknowledging that perhaps *we* should be stepping back until we feel we have more clarity to address the situation appropriately).

Maintaining this kind of emotional labor can be incredibly exhausting. We must acknowledge and accept that being perfect is unattainable, and more importantly, not a requirement for being a great leader. What *is* important is people can relate to your human side.

People gravitate to others with whom they share a kinship [15], and being able to identify this feeling of kinship is one hallmark of an effective community manager. If your members see that you possess qualities they can relate to, they can more easily empathize with you. Ironically, we often emphasize the significance of practicing empathy for our members or team, but it's equally important that our members demonstrate compassion and gratitude towards us too.

As everyone on a team or in a community nurtures this empathy, they will gradually deepen connections and trust between them, which in turn can help them establish an informal social support network. This network can be a conduit for promoting the importance of self-care, creating judgment-free zones, or providing safe havens to individual members (including yourself) for emotional reflection, airing frustrations, or sharing workloads.

It is inevitable that some members will expect you to adhere to the pretense of being the all powerful, infallible captain of the ship, but with an effective self-care routine and the backing of the members within this social support network, you'll feel more confident in your ability to handle those stressors. You'll also understand that your vulnerabilities are what makes you a better community leader.

8.4. Types of self-care

Everyone will prefer different self-care techniques and strategies, depending on their moods and circumstances. To be effective, self-care requires regular and conscious cultivation, so it's important that we view self-care not only as a reactive choice but also as a means of alleviating the stresses of everyday life.

In general, however, a number of different self-care types can satisfy our basic need to promote a healthy and happy mind and body. These are: *physical*, *mental*, *spiritual*, *emotional*, and *social*.

Next, we'll explore each of these types in more detail. But remember: we should be aiming to practice a *selection* of activities of *all* these types if we're going to provide ourselves a healthy life balance and respond adequately to all types of stress.

8.4.1. Physical self-care

Physical self-care is usually the self-care we perform at a minimum, often subconsciously: feeding, hydrating, sleeping, and exercising.

However, we often find ourselves neglecting these necessities for the sake of work (enduring frequent all-nighters, for example, or forgetting to eat lunch every weekday). Keeping ourselves nourished helps us maintain bodily health. Getting into healthy physical self-care routines also helps us take regular breaks from our work—and our work *environments*.

Physical self-care might include activities like:

1. Maintaining a regular sleep routine
2. Eating a healthy diet
3. Taking a nap
4. Getting a massage
5. Going for a stroll
6. Stretching
7. Doing yoga (or other forms of exercise)

8.4.2. Mental Self-Care

Mental self-care is the act of stimulating our mind with positive and purposeful thoughts to help reduce stress levels.

These are doing things that keep the mind engage at an intellectual level on topics that interest you or help de-clutter your thoughts to re-organize them.

Mental self-care is often less tangible than other types so it can be more difficult to see an immediate benefit.

However, with consistency of exercising mental self-care we will see it's benefits shape and form healthy attitudes towards others aspects of our life as we will be more inclined to be mentally satisfied.

A few examples of mental self-care:

1. Reading a new book or article

2. Trying a hobby or interest
3. Writing a list of goals
4. Solving puzzles
5. Organizing or cleaning out a space in your room

8.4.3. Spiritual Self-Care

This type of self-care often gets wrongly associated with being solely about religion but it can be applied to everyone whether you’re religious, atheist, agnostic, or otherwise.

Spiritual self-care are activities that nurtures the connection between you and your soul, providing you a deeper sense of meaning, or understanding of the universe. The word soul is merely a representation of the entity or uniqueness you feel embodies you, this can also be your inner spirit, energy source, or another reference.

A few examples of spiritual self-care:

1. Volunteering for a cause you care about
2. Meditating
3. Spending time in nature
4. Praying or attending religious service
5. Determining your most important values or morals
6. Considering your significant relationships
7. Discovering new forms of spirituality and religion

Regardless of the different types and activities of self-care we perform, the aim is to help us in a constant and sustainable way, to fight off and defend us against the negative effects of our role. By ignoring our physical and mental well being we will be more likely to succumb to the stress and fatigue leading us towards more dangerous chronic illnesses and syndromes, like burnout.

8.5. Burnout

What exactly is burnout? The WHO definition of burnout is:

> "Burnout is a syndrome conceptualized as resulting from chronic workplace stress that has not been successfully managed."[16]

Burnout can affect us all and in any occupation, however it seems more prevalent in roles that are mentally and emotionally draining for extended periods of time. This is common due to the prevailing norms within those roles of being selfless and putting others first [17]: going the extra mile to maintain a happy and content environment or atmosphere either for the client or within a community.

It is also appearing more and more within the tech industry.[18] This increase has been attributed to the seemly accepted 24/7 work mentality and competitiveness of the industry, leading to workers involved in technology, particularly software development, to becoming overwhelmed and mentally exhausted to the point of risking their health.

We should highlight that work related stress and burnout are very different, and in cases some amount of stress can provide a source of motivation but only if it is manageable and for a temporary period of time. When occupational stress is long occurring, seen as chronic, affecting the overall well being of ourselves, this can develop into what is termed as *burnout*.

8.5.1. Look out for symptoms

Burnout is extremely hard to detect as not only is it subtle and progressive, but it is often misdiagnosed as the earlier, more temporary, common work related stress. This is because the two are similar until it becomes too late and has developed into a much deeper and harder problem to treat.

Psychologist Herbert Freudenberger has released multiple books and articles since the 1970s regarding his research of the possible causes, implications, and affects of burnout. His work [19] has helped to define the different symptoms and thus the phases of experiencing burnout.

Perhaps you recognize several of them in yourself; perhaps you recognize only one or two. It's not always easy to see the signs since not only do they gradually occur over time, but also hide behind our own denial of something being wrong.

Exhaustion

Loss of energy and accompanying feelings of weariness are usually the first distress signals especially if you naturally have high energy levels.[20] However, be careful not to push yourself harder if you do find yourself struggling to keep up with your usual round of activities. Doing so will only exacerbate the problem.

Similarly to our emotions, our energy also affects others around us. We tend to fuel our energy by achieving our goals and reaping the rewards, thus sharing that with others. If we are unable to attain rewards due to the lack of energy levels then this feeds into a vicious cycle.

The things that once excited us, like leaving a meeting fired up to accomplish an objective, have now become mundane and seen as an excessive use of our already depleting energy. You may not see the lack of accomplishments, like others do, because you see less and less significance in obtaining the rewards and blame your tiredness on your increasing workload.

Detachment

We usually demonstrate a sense of detachment or apathy as a self-protective device to help ward off emotional stress or pain. When we begin to feel let down by situations or those around us, whether that is the team, community, company, or even ourselves, we are temped to down play their importance; "I don't care, it wasn't important anyway," and move away from the things that used to involve us. By doing so we are depriving them the power to affect us negatively, however, this also blocks their ability to positively affect us. This can lead to loneliness and isolation.

Boredom and Cynicism

Once you’ve become more detached from the things that excited you, you find it increasingly hard to remain interested in what’s going on around you. You begin to question the value of your activities, your relationships, and perhaps the bigger aspects of your life. This can lead you to becoming skeptical or even suspicious of other people’s motives and causes.

Impatience and heightened irritability

People who have high energy levels also usually have a characteristic of being mildly impatient, whether it is with others or with themselves, due to their ability to perform things quickly to then progress onto something else. However, when experiencing burnout, the perception becomes that we need to over-accomplish things and thus so does the impatience to do so. This impatience can spill out over to others as irritability with everyone around them. Things that were once trivial and minor become huge obstacles often with the blame pointed at others creating it rather than ourselves.

A sense of omnipotence

We don’t start off feeling this way about our role, but often when we are overwhelmed with our workload we can default to a sentiment: "No one else can do this, only I can."

This sort of statement is often an attempt to justify the over exertion of the effort and applying value to it while other areas of our workload are failing. It’s that grasping for control when things are becoming out of control.

Rest assured that indeed others can perform those tasks, though differently and maybe not to the same degree of excellence you may have done but it could be a situation that doesn’t always require excellence. This type of egoism is more often a hindrance to progression and the initiative of others.

A suspicion of being unappreciated

To counter-balance our lack of energy we often increase our efforts, but this doesn’t necessarily reflect good results. However, we don’t acknowledge this, we only see the effort expended. We can then begin to feel like we’re being less appreciated from others in the team or the community as a whole. "Can’t they see all the hard work I’m doing, staying late at night?" This feeling can lead to being bitter and angry.

Paranoia [21]

Leading from the signs of feeling unappreciated to feeling as though the world is against us. When things go wrong, but we are unable to understand or see why, we tend to seek out a target, not ourselves, to blame regardless if there is little merit in the accusation. Often the person labeled as the culprit becomes the target of our frustrations. This can be team members, friends, or even family.

Disorientation

Disorientation is when we feel we've become separated from our environment and understanding of what is going on around us. Discovering yourself in a situation that you didn't become aware of, or realizing that you previously understood a concept but now do not. We see ourselves starting to forget things easily and our concentration span deteriorates leading us into more confusion and agitation, fueling the other symptoms like paranoia.

Psychosomatic complaints

This is not to be misunderstood to imply those experiencing signs of burnout are not feeling physically sick; they can and do. But it does highlight that with prolonged stress physical illness symptoms appear as a secondary symptom to the cause, like lingering colds, backache, headaches, etc. Sometimes these illnesses mask the deeper more emotional stress that we feel but we feel more comfortable taking a sick day instead of actually acknowledging the mental stress.

8.5.2. Burnout cycle

Freudenberger and his colleague Gail North [22] later categorized the consequences of these symptoms into 12 phases of one developing burnout syndrome [23]. Similar to the symptoms, sufferers may experience episodes in multiple phases, not in sequential order, and for any length of period of time.

1. **A compulsion to prove oneself**: desire to prove oneself, to have impact on one's peers, initially seems beneficial until this desire turns into obsession.
2. **Intensity (Working Harder)**: compulsion becomes misconstrued as dedication and commitment. This can appear as an unwillingness to delegate work, for fear of losing perfect control, or working harder and longer.
3. **Neglecting their needs**: work begins to dominate and subtler duties and pleasures are viewed as unnecessary like sleep, eating healthy, etc.
4. **Displacement of conflicts**: conflict from others are considered meddlesome and seen as a threat. Coping mechanisms are put into place to dismiss problems and these can manifest into physical breakdowns.
5. **Distortion of Values**: focus on work only, values are distorted as well as relationships. This leads to them being dismissed or abandoned.
6. **Denial of Emerging Problems**: mechanisms to defend oneself against the impact of life and in turn their demands. Develop inability to tolerate ambiguity and become non-receptive; projecting the anxieties and insecurities externally.
7. **Withdrawal**: Become detached from our emotions and from other people. Often "escaping" through television, books, or other means like alcohol/drugs.
8. **Odd Behavioral Changes**: friends and family identify increasingly obvious changes in behavior like attitude, language, or physical activities.
9. **Depersonalization**: viewing the needs of one's self and others are now significantly undervalued and dismissed.

10. **Inner Emptiness**: feelings of hollowness and uselessness. There is a desire to replenish but are usually just quick wins, or false cures and ultimately unfulfilling.
11. **Depression**: feeling of being hopeless and joyless. Despair and exhaustion are primary feelings and the overwhelming desire to escape.
12. **Burnout Syndrome**: suicidal thoughts, physical, and mental collapse leading to life threatening situations. Immediate professional medical help is imperative.

These distinctions help us to identify the deterioration in either our own, our team's, or community member's activities and their attitudes towards themselves and others.

It's important to be self-critical and pierce our disillusion that everything is fine – it usually isn't and it won't "just work its way out."

8.5.3. Causes of burnout

We've identified the devastating effects of burnout now lets explore the possible sources for these symptoms within our role or even within the community.

We earlier described that burnout is a combination of many factors but a recurring element is the realization, subconsciously or not, that we don't feel our work is providing us the same sense of reward and purpose as it had once done before.[24] Rewards don't always equate to money or status but can simply be the deeper satisfaction and pleasure in the adhering to one's values and achieving happiness.

Lack of Control

To feel a sense of accomplishment and ownership of a task, a role requires a suitable level of autonomy to achieve this. If we have the inability to influence our decisions or don't have access to appropriate tools or resources, this can lead to the de-motivating feeling that our work and effort is not being appreciated enough or we are not trusted enough with this responsibility.

Lack of control can also manifest when dealing with other peoples' emotions. Although we can encourage and try to direct our members to adhere to our community's code of conduct or a preferred course of action in a conflict, we evidently cannot remove their willfulness. We must only pre-empt their next move no matter how disastrous it may be. This can lead to the feeling of being constantly in firefighting-mode and not accomplishing anything.

Unfairness

Unfairness within the role can be viewed as a number of different things that attribute to feeling powerless or being disrespected. Either you or others are treated unfairly, such as: office or community politics that create a culture of favoritism, lack of transparency in the top-down decisions, or a disproportionate amount of workload is allocated to you.

Insufficient Reward

You feel unappreciated, taken for granted or simply not satisfied in your role. Rewards don't always need to be monetary but often this is the first thing to come under our scrutiny when the workload increases.

We also need social rewards where we gain the recognition from others. A lack of recognition can be from the company itself not appreciating our worth, the team's lack of respect towards us, or from the community not seeing all the "behind the scenes" activities we perform.

Intrinsic rewards are also important to maintain a healthy perception of our role. This is where you take the self-acknowledgment of doing a good job and feel accomplished. When we feel we aren't living up to our standards we begin to feel disappointed and become de-motivated.

Sometimes we feel unsatisfied because we have a conflict of personal values with the company or project we work with. We are often asked to relay and even promote the decisions of the company to the community and these may not align with our own personal values. This can be seen as self-betrayal to our morals and build up resentment towards the company.

Work Overload

Probably the most common experience contributing to burnout is the over-burdening of one's workload [25], whether from our own doing or by someone else. This can occur when the quantity of work and expectations exceeds the amount of time or resources available. We may feel that most other employees expect work assigned to us is "urgent", when in fact it may not be. It's important to maintain boundaries and stand your ground to resist an ever increasing list of things to do.

Lack of Community

It goes without saying that community is extremely important; it fuels the purpose of the role as a source of motivation and companionship—a sense of belonging to you as a person. However, if this becomes stagnant, overwhelmed with toxic members, and feedback is non-existent, this can make the job feel stifled.

8.5.4. Preventing/treating burnout

If you feel yourself or anyone else succumbing to burnout then the most direct approach is to take a break from the source of the stress, which is more often work itself, and reflect on the more acute causes of your burnout.[26]

Use your holiday time

Don't be afraid to utilize this time and don't feel guilty either. Using your holiday does not demerit your dedication to the role, neither does it mean that everything will fall apart while away. Use this time to concentrate on yourself, and what gives you pleasure in life.

Spend time with those you care about

Re-kindle your social relationships, they have probably missed you as a result of the developing burnout. Talk through how you're feeling and enjoy your time with them so it is overall a pleasant experience.

Try to generally stay clear of negative people in your life. This could mean letting them disappear from your social network, or limit your interaction with them. Remember, other people’s emotions can affect us both positive and negatively.

Re-evaluate priorities

Identify what is important to you and reflect upon if your current lifestyle, or work life balance mirrors that. If they don’t, then prioritize what you wish to enjoy more, block out time in your schedule, and commit to it.

Also evaluate your options and consider what the next steps would be to resolve the stressors you have. This could be coming to a solution or compromises with your line manager to reduce workload or other concerns you have. There may be a point that the only way to remove certain stressors in your life is to leave your job to improve your health.

Practice self-care

Take the time to commit yourself fully to what ever self-care activity you want to enjoy and do it. Try and practice self-care daily, detaching yourself from as much work as possible and devote yourself to some "me" time.

Seek professional help

If all the other options have little or no affect on your physical or mental well being, or you feel you require immediate assistance, then do seek professional help as a matter of urgency.

8.6. Work-life balance

A healthy work-life balance is having a clear distinction between our personal and work lives without allowing one to dominate the other. Both are equally important and neither should be undervalued. We can find ourselves in unhealthy mindsets when forced to be stuck in either one extreme or the other, withholding an important sense of purpose and enjoyment from that part of our lives.

It has also become more difficult in this day and age to detach ourselves physically from our work life. Technology has provided us such a convenience that we are in almost constant connection to it, and thus in connection to our online communities. It is a common place to check emails at all hours, or respond to members of communities on our social media network.

As well as this physical difficulty we may also have the emotional difficulty of switching off from work. We can feel it’s a requirement of our role to be available 24/7 and be responsive as a reflection of a caring and active community. This is often not the case, and in fact is counter productive in building a sustainable community and providing quality interactions with our members. Leaders don’t need to respond to all messages to be great.

Each person’s work life balance is different with each their own prioritizes. This is where self-care activities play a big part in establishing the distinction between work and personal life. Make a clear differentiation of what you view as work, like answering community requests or emails, arranging calls or meetings, etc. and the hours you aim to dedicate to work; anything outside of that communicate to yourself and to others that is your personal time. By dedicating a consistent and explicit downtime, we begin to develop a habit that our body and mind anticipates and begins to look forward to, making it easier to develop and maintain a good habit.

8.6.1. Addiction

Work addiction, often referred to as workaholism, can affect anyone who is deeply embedded in an online community and often justifies their extensive work hours as commitment to the project. The inability to stop is often driven by the compulsive need to achieve status and success, or in some cases to escape emotional stress. Work addiction can be a vicious cycle where the feeling of achievement is an addictive "high" at the cost of our mental and physical well being, often not noticed until too late.

Work addiction, like other addictions, is difficult to acknowledge there is a problem to begin with. People suffering from work addiction are often in denial, convincing themselves work is a pleasure. Eventually this over compensation of effort and time, neglect of personal relationships and well being, leads to the inevitable experience of burnout.

It’s important we develop a healthy relationship with our role itself without feeling the need to be on the pedal at full gas. Try and assess what truly drives your motivations to achieve and does this require you to be online the amount of time you are. Do you find that you feed off external praise as form of validation of your work? Do feel that if you walked away from the community it would fall apart? Identify those moments of pleasure, whether it’s completing a task, or receiving a compliment from a community member or boss, and evaluate whether they are needed in the same doses you are currently experiencing them at.

We can also find that this need to achieve is a reaction to a heavy workload from the lack of resources within the team trying to prove to others the value the role and team brings to the project or company.

Reconsider these goals with the aim to reduce your workload. Are they achievable and maintainable with the current resources without sacrificing quality and a good work-life balance? If they aren’t, then consider prioritizing and communicating the most impactful goals that the team can achieve. Delegate any other tasks to suitable members or establish more flexible timelines, and anticipate time for possible firefighting as part of those deadlines.

Not only does this help to set reasonable expectations for the team members to achieve, but also promotes that a healthy work-life balance is an integral part of their schedule. This predictable schedule also helps you to provide better forecasting to the company or community.

8.6.2. Maintain boundaries

When reflecting upon our work-life balance, it is important to establish clear boundaries between the two. As we've said earlier, due to our nature of work, we find ourselves participating within the community, and this begins to eat into our personal time, leaving nothing else. This is tolerable only on a temporary basis and only when we are required for an intervention, but this should not be the norm. Boundaries help us establish where our work ends, and pleasure begins. We're not saying that work isn't pleasurable, but having a variety of activities other than work helps stimulate our minds and provide alternative creative outlets.

These boundaries can also help the community acknowledge and accept your expectations of them as well of what they can expect from you. Be as transparent as possible by defining your available hours, and an escalation process for obtaining help outside of those hours. Highlight the importance of documenting community processes so members feel more informed on what they should do in incidences, with or without requiring assistance. The aim is to establish a consistent schedule, and to have the team and community respect it. Although they may not do so on every occasion, you will be able to use your boundaries to help prevent the feeling of guilt as you begin to embrace personal time as your own, as well as respecting others.

Of course if there are any serious incidences that require your intervention during down-time, ensure you put into place mechanisms for the team to handle them rather than you being the only one who 'can handle it.' These mechanisms can be an escalation process or a team effort to respond and review the response collectively. This helps encourages the mentality that everyone can lighten the load, especially when it eats into yours and their personal time.

Maintaining personal boundaries is also extremely important as well. Our role often asks us to help members with their workload as well as interpersonal communication matters between themselves and other team members. But we need to be aware and recognize that we can't solve every interpersonal issue or conflict–sometimes we just can't become too involved.

As much as we don't want to admit it, we must respect that we are not skilled or obligated to practice therapy if we feel it is required for a particular member. When the conversations or observations become more apparent in that direction, then aim to persuade them to seek medical or psychiatric help. Our role is to aid members, but there is only so much we can achieve from our position and that is OK.

It can be beneficial to partake in mental health training for you and your team to learn how to handle situations involving members in the community or team. This can help you apply a suitable process to follow upon if someone is beyond your ability and responsibility to help them.

8.6.3. Sustainability

Sustainability is an extremely important goal to have for a community, often seen as a contributing factor to the project's own success. This should always be at the forefront of our minds when developing tools and processes for the community, with the aim for the community to become self-reliant, self-driven, and empowered. But there is a lot of work to be done to achieve this, and we need to ensure we and our team are able to keep up.

Things become unsustainable when we have set unrealistic expectations either upon ourselves or on the community. When it comes to ourselves we can underestimate our project timelines because we have attributed our motivation as part of the estimation: the drive that will get us over the last hurdle. Motivation is not an unlimited supply, and can fluctuate drastically due to external and internal factors. Try to extract motivation as a factor—although you may feel extremely excited about a project, don't let that cloud your judgment on how long a project will take to complete. If not, you may see it negatively affecting your work life balance.

We tend to also inaccurately assume the motivation of others in the community. By definition community member are volunteers and, yes, we are fortunate to have those exceptional members that go above and beyond what is required. However, we should not expect the same of all, in fact we should expect delays and anticipate them.

By beginning to form clear boundaries, reduce your workload expectations, and improve estimations, you start to deliver on realistic schedules. Imagine you achieve a task within a week, rather than it taking triple that amount of time, because: you identified it as a priority; delegated other lower tasks to the team (or set the expectation it wouldn't done at all); only worked within your allocated time; and were refreshed from recharging your mental well being with dedicated offline time. This combination of activities and processes was key to achieving success, thus triggering the event of providing and receiving continuous rewards, and helping towards reducing the probability of members developing burnout.

The only thing that is ever consistent is time, so be aware that you may find the same rewards you gave yourself and others, change over time. Take time out to frequently reflect on what drives you and your community, positively review how much you have progressed, and assess what resources you have to adjust project goals accordingly without interfering, if possible, with a healthy work life balance.

8.7. Self-reflection

8.7.1. Through the looking glass

An important aspect of being a manager or leader is to provide good and constructive feedback to those that are on our team, as well as the community as a whole. We understand that feedback from upper line managers/other leaders and those that report directly to us is extremely important to understand their perception of us as a person and our activities representing them: if they truly reflect our efforts.

Retrospectives are now almost integral in software development teams to continuously improve individual or team performance and morale, and identify problems that need solving. However, we find we don't often do a retrospective for ourselves, with ourselves.

Introspection is the examination of one's own conscious thoughts and feelings. This can refer to the mental state or in a spiritual sense, one's soul. Self-reflection, introspection, and self-care are all intertwined with the aim to promote and sustain a positive direction for mental growth and development.

Introspection is extremely important for ourselves to evaluate our purpose and happiness we get from our actions, thoughts, and behavior. Since work is a big part of our lives, we want to ensure our role within the community and at our company aligns with our values. Or else we will find ourselves becoming more and more dissatisfied by the role's insufficient rewards.

But first we need to know what our values are, what qualities we enjoy out of the role, and the characteristics of the people we love to work with.

Take some time to truly answer these, as gaining this self-awareness does not happen over night. Use these answers to help you reflect on how you feel when you do the things you do, both positively and negatively. Journaling is often a good, yet simple, practice you can do to clarify your thoughts.

Practicing self-reflection can be difficult to begin with due to previously discussed inner restrictions we place upon ourselves as community leaders: the need of being invincible; distorted perception of our worth; and lack of visible support. However, in creating a routine of introspection and self-reflection as part of our self-care, we will begin to exercise more control over our emotions: have inner clarity on our long term goals, and ability to identify more solutions-focused activities rather than the previously emotionally driven ones.

8.7.2. Tackling imposter syndrome

This term was first defined by psychologists Dr Pauline Clance and Dr Suzanne Imes [27] in the 1970s as the internal experience one feels, despite overwhelming amount of evidence proving otherwise, that they are incompetent and that their success was a product of luck or fraud within their field of expertise.

Often those that experience impostor syndrome have a hard time internalizing and accepting their success by minimizing positive feedback and comparing other's work to their own. This more frequently happens if we have started a new job, taken on new responsibilities or roles, or returned from a recent career break. In order to compensate for this chronic self-doubt we begin to work late, procrastinate, or try to justify our position in unnecessary ways.

Dr Valerie Young [28] further categorized these types of flawed thinking of what sufferers believe it takes to be competent into the following subgroups:

Perfectionist

Perfectionism and imposter syndrome tend to go hand in hand. When a perfectionist doesn't achieve their unreasonable high standards they question their abilities and thus if they deserve to be in the position they are in. If they do successfully achieve their goal, there always seems to be that unattainable objective they expected to have reached or knowledge they expected to have but didn't.

Natural Genius

These sufferers feel that the natural ability to achieve a task is a direct correlation to their competence. If they take a long time to master something they feel that it has less merit. Not only do they have high standards but they also have to complete it without breaking too much of a sweat.

Soloist

These are those that shy away from asking for help because they fear that would expose them for who they believe others to see them as – a fraud. Although being independent is good, it can lead to sub par results without acknowledging that two heads are often better than one.

Expert

People with this complex of impostor syndrome often dismiss their success because they don't know everything there is to know about the topic or role. Often these people dislike to be put on the spot in case there is some aspect they were unaware of and thus exposed as a fraud.

Superhuman

Usually these people often over compare themselves to others in their industry, the seemly high achievers, and push themselves to work harder and longer to measure up to them. They also tend to heavily rely on external validation.

Since our role as community managers is relatively new and less established than other roles within the tech industry, we can find ourselves struggling to easily define and confirm our decisions due to the lack of expertise and documentation in this field. We can find ourselves feeling more aware of being identified as a fraud especially when the company or project has never had a community manager before.

However, there are ways to help keep impostor syndrome in check and increase your self confidence.

Celebrate Successes

Frequently write down our successes and enjoy them. Journaling is a good way to have comparisons from earlier successes and how they lead up to our current ones. Include our own account of successes but better yet include testimonials from others, be it from community members responding to our thread posts, or colleagues praising our work. This will help support that feeling that we are contributing value in our role and others confirmed it.

> "We don't attach to people or things, we attach to uninvestigated concepts that we believe to be true in the moment"
>
> — Byron Katie

Change your perspective

We are hindered by our fear of being exposed as a fraud, but usually we don't have the proof that confirms that is the case. We often wrongly assume and interpret actions of others as a direct cause and effect to things we have done or said. This is because we are viewing the situation from our perspective and only from ours.

Concentrate on what value your work brings to the subject or community and visualize that success. Imagining good things happening can give you the confidence, and motivation, to commit to the task at hand and overcome the fear.

Working in progress

We are always learning, improving, and progressing. Treat our successes as continuously developing projects, adding refinements into each iteration. Not only will we be able to record multiple successes but also help acknowledge that perfectionism is impossible and mistakes are opportunities for better learning.

8.7.3. Network of support

We understand the power of a community, the ability to bring people together and with the right directions—and a whole lot of love—we can move mountains. So why do we feel we can't have the same mentality toward helping ourselves?

During stressful and tough times, whether it's just a bad day, or more chronic episodes of illness, research has shown that having a strong—though not required to be large—social support network is beneficial to our well being [29] Without a social support network it can feel lonely and isolating which can lead into further depression and anxiety.[30] Often it's our social support network, even if we don't think we have one, that first spots there is a change with our behavior before we do.

A social support network is made up of friends, family, and peers.[31] Although this is different from a support group, which is more formal and often prescribed, a social support network is something we can develop as part of our community and team structure to help tackle stress, and promote self-care.

Look towards those around you who you have a good relationship with and you feel you can confide in them. When you are feeling stressed or want to simply vent your frustrations, come to rely on your social support network to let go in a safe and healthy way. This unburdening of tension helps untangle your emotions, seek clarity on an aspect of decision making, or just lightens your mood by the sheer enjoyment of speaking with them.

We may find that those within the community, whom we spend most of our time with, grow to be included in our social support network and that each individual provides us with a unique form of support to help in different ways in our lives. But also remember that we should also serve as a form of support to others.

The more education and communicating with our members about the benefits of self-care, the more likely we will see it being practiced and encouraged by others. This in turns helps create a more caring and accepting atmosphere in the community. Education can be in the form of discussions promoting self-care, celebrating mental health campaigns [32], adding to the community guidelines when on-boarding team members to speak to the team if their workload or other aspects is affecting their health [33][34], or organizing training for team members on mental health awareness.

If you see a member on the team or community showing symptoms of burnout then reach out to them and let them know you are concerned for their well being. Identify that you are there to support them and more often they will respond positively and work together to alleviate their stress.[35]

However, it is important to make clear here that if we feel that we are unable to assist a community member's emotional stress beyond our role's capacity, then encourage that they seek professional health advice immediately. We may find ourselves feeling guilty we are unable to provide support, but we need to remind ourselves that we are not professionally trained and thus could provide—though well intended—ill advice.[36] Remember that other emotions affect those around them including how member's stress can affect ours.

Similarly in our own direct reports' one-to-ones ensure you also have regular one-to-ones with your line manager to highlight any problems you have achieving your workload or effecting your well being. Be as direct as you are with helping others, as you are with yourself.

8.8. Resources

- **High-Octane Women: How Superachievers Can Avoid Burnout**

  by *Sherrie Bourg Carter Psy.D*

- Burnout, Your first ten steps

  by *Amy Imms M.D*

- **Burn-out : The High Cost of High Achievement**

  by *Dr Herbert Freudenberger and Geraldine Richelson*

- **Women's Burnout: How to Spot It, How to Reverse It, and How to Prevent It**

  by *Dr Herbert Freudenberger and Dr Gail North*

- **The Secret Thoughts of Successful Women**

  by *Dr Valerie Young*

- **The imposter phenomenon in high achieving women: Dynamics and therapeutic intervention.**

  by *Dr Pauline Clance and Dr Suzanne Imes*

[10] World Health Organization, website

[11] The Community Roundtable, 2019 State of Community Management Survey

[12] Sherrie Bourg Carter Psy.D, Emotions Are Contagious - Choose Your Company Wisely

[13] Principles of Social Psychology, The Role of Affect: Moods and Emotions

[14] Alex Budak, In-Control vs. Under-Control Leadership

[15] Psychology Today , Why Do We Like People Who Are Similar to Us?

[16] World Health Organization, website

[17] Herbert J. Freudenberger, Staff Burn-Out

[18] Team Blind, Close to 60 Percent of Surveyed Tech Workers Are Burnt Out…

[19] Dr Herbert Freudenberger and Geraldine Richelson, "Burn-out : The High Cost of High Achievement"

[20] Maslach, C., & Leiter, M. P. (2008), Early predictors of job burnout and engagement. Journal of Applied Psychology, 93(3), 498–512

[21] R Bianchi, L Janin Burnout, depression and paranoid ideation: a cluster-analytic study

[22] Dr Herbert Freudenberger and Dr Gail North, "Women’s Burnout: How to Spot It, How to Reverse It, and How to Prevent It"

[23] Freudenberger’s 12 stages, Freudenberger’s 12 stages

[24] Adeva What causes employee burnout in the tech industry

[25] The American Institute of Stress: Survey, The AIS Workplace Stress Survey

[26] Pyschology Today, When Life Loses Its Meaning: The Heavy Price of High Achievement

[27] Dr Pauline Clance and Dr Suzanne Imes, "The imposter phenomenon in high achieving women: Dynamics and therapeutic intervention."

[28] Dr Valerie Young, "The Secret Thoughts of Successful Women"

[29] American Pyschological Association, Manage stress: Strengthen your support network

[30] Siv Grav, Ove Hellzèn, Ulla Romild, Eystein Stordal, Association between social support and depression in the general population: the HUNT study, a cross‐sectional survey

[31] Mayo Clinic, Social support: Tap this tool to beat stress

[32] Mental Health Foundation Mental Health Awareness Week

[33] Ubuntu Ubuntu Burnout

[34] Ubuntu Burnout Help, Ubuntu Burnout Help

[35] Jono Bacon, *Detecting and Treating Burnout*, "The Art of Community"

[36] Chartered Management Institute How to Talk About Depression at Work