# MAINTAINERATI *BERLIN* 2019

## Session Notes

# *MAINTAINERATI BERLIN 2019*

## Session Notes

Prepared by the Maintainerati Foundation Board





Revision 1, 26 March 2020.

# *TABLE OF CONTENTS*

# *SESSION NOTES*

This document contains notes from each of the 29 discussion sessions held at the Maintainerati Open Space in Berlin on 24 May 2019. It is a companion document to the main report, *Maintainerati Berlin—Event Report*, available on the Maintainerati website.

During each of the 29 discussion sessions a dedicated person took notes using a pen and paper or a computer. The sessions were not audio recorded. Each session ran for one hour, and was assigned to a dedicated note-taker whose role was to capture as much of the conversation as possible. They were also asked to provide us with their own interpretation of the conversation that took place, thus producing an initial analysis.

The Research Team then cleaned up the notes to make reading and understanding them easier. We took care not to change the voice or intent of attendees' statements as recorded by the note-taker. As much as possible we omitted any details that might identify individual participants. Each summary includes "Takeaways", observations and advice given by the season's attendees.

Each session's conversation was unconstrained, and often took unexpected twists and turns, so don't expect a clear, concise narrative for each session. Rather, think of these notes as data to be examined and further analyzed.

We are making these session summaries freely available in the hope that maintainers, researchers and policy-makers can use them to better understand the problems faced by communities developing digital infrastructure. We have provided our own analysis in the Event Report, but we encourage the open source community to dig deeper on their own.

# *Avoiding burnout*

This session was one of the most popular ones of the day. The group discussed what causes burnout and shared tips on how to deal with it, based on their own experiences. Overall, the attendees were proud of their projects and wanted them to continue, but many of them had experienced burnout and were still struggling to achieve life balance.

Why is burnout so common? Attendees mentioned various reasons. Some felt trapped into maintaining their projects, since people depended on their leadership for the project to continue. Some found themselves doing activities they had not anticipated: they got into their projects through coding, but now spent their time managing their project. Many enjoyed this new-found management role, but struggled to meet users' and contributors' expectations.

For example, if a maintainer asks someone to "submit a pull request", the person being asked might see this as rude. However, no one thinks twice asking a maintainer to fix a bug in a project that they are maintaining for free.

There are also technical and cultural issues around communicating the state of issues and bugs. Historical information is also sometimes lost as to why things work the way they do. Getting folks to file "good" and "actionable" issues can be a big challenge. Rude messages and issues are mentally and emotionally draining.

The sheer volume of work also presents a challenge. There is always something that desperately needs to be done. Positive input can be very much welcome, but on the other hand, receiving 1,000 "thank you" messages also increases workload. Also, the US work culture (50h+/ week) and glorification of workaholics supports burnout.

One problem with running projects is a lack of common knowledge:

> "If you don't create common knowledge it can create a mess."

One academic project discussed did not practice regular recording of common knowledge in a central place, and this seemed to be a core reason for their management problems.

People are concerned that they are rude to people when they defend their own time by telling them to be more explicit or to make a PR. But

it's also demotivating if people open issues to update documentation or report trivial bugs.

There seems to be a constant struggle between keeping one's sanity by ignoring or being strict about request and feeling guilty about not meeting people's expectations:

> "No matter what you do, you'll always find people who don't like it."

There's also a difference between "people who just want to use your stuff" and "people who want to contribute". One participant mentioned that he gets requests from big companies "who should know better" than to ask for so much. Another one mentioned that "academia doesn't give back to the community because they don't have time".

It can be difficult to let others fix issues because they might miss the context and evolution of the project and are not familiar with why something is implemented or behaves in a certain way.

Participants also acknowledge that the whole "issue" terminology has a negative connotation – it leads to people reporting only broken things and complaining about missing features. It's "mentally draining".

Participants also shared their experiences about going from a contributor role where they "just wanted to fix something for myself" to a project manager role, where they have to communicate and align people. One person described how he found someone to help him out with a project, and said with a smile on his face, "I can finally go back to just coding". But it's also difficult to find someone you can trust, and so "you feel like you can't step away".

All these things make management difficult and contribute to burnout and the feeling that "it's not fun anymore". The group discussed how burnout rates are higher among people who became maintainers out of necessity, because there was nobody else available to take on the role.

## Takeaways

- As a maintainer, you need to realize that your time is more valuable than that of members of the community.
- Have clear guidelines for yourself.

- Collaborate and delegate!
- Don't be afraid to push back on folks to prevent you from becoming "the janitor" of all things, rather than an engineer writing code.
- Have your users create your repo cases.
- Maintain a separate work phone.
- Switch from push to pull notifications or disable notifications.
- Keep your maintainer email separate from your personal email.
- You don't have to fix all the issues. You don't have to fix ANY of the issues.
- If people ask you for things, it's not rude to ask them to put in an equal amount of work.
- Creating templates for issues and contribution guidelines makes it emotionally easier to close issues for those that don't follow the guidelines / create reproducible steps.
- Use bots to close issues that aren't properly filled out or to close issues that are over X days (people <3 StaleBot). People seem to be more generous with bots than humans, this helps to reduce conflict.
- To help others make meaningful contributions, communicate the context of the project and why specific decisions were made.
- Don't hesitate to open issues asking for help, such as in looking for another maintainer or for help reproducing issues.
- Start new projects with at least one other person.
- Your primary long term job as a maintainer is finding your replacement. This removes the pressure by giving back a sense of control.

# *Communication / abuse*

This session had five participants. It started with the host outlining what they thought of as the three communication channels:

Written communication, which is asynchronous.

Documentation as a stagnant place to reference information.

Synchronous communication, either face to face or online.

There is pressure to do stuff on a lower platform (mobile), which is not an ideal place to interact.

The first participant who spoke stated that their reason for joining the conversation was because they had learned that on a number of instances their way of reacting to contributors was too aggressive. They learned this when their aggression was called by somebody who they respected as a contributor.

A second participant in the conversation then shared a story about an interaction that occurred when they created a repository in an effort to start a new project. A particular developer on open source projects, who also happened to be a co-worker, pointed out that they had failed to sign a license agreement when they started the repository and told them they needed to do so. The participant saw their co-worker as being aggressive because they jumped on them while they were in the process of signing such an agreement. They felt like they were not given a chance.

The group discussed how written communication is difficult and it takes effort to frame things and write them out. Documentation can also sometimes be used as a weapon. Tone can be hard to convey online, and so sometimes in-person conversations are helpful to identify what each party meant in a written conversation. However, the participants in the group disagreed as to what counted as aggression and what did not. The group also briefly discussed maintainer abuse by contributors.

Attendees noted that GitHub have formalized some common norms, but attendees raised questions about how effective these are for those not already familiar with the medium. Responding too quickly can also be outside of the norms.

## Takeaways

- Create a bot to welcome contributors.
- Provide maintainer's guidelines as well as contributor's guidelines. [editor's note: these do exist, but it seems not everyone is aware of them, or perhaps people find them inadequate]
- Going over PRs together can help reduce unintended distancing or gatekeeping tone in code reviews.
- In open source discussions, maintainers should be introspective about how they are contributing.
- Try not to assume that people know what they are doing: everyone has to learn sometime.
- On the flip side, it's also a good idea not to assume that people don't know what they are doing: they may not be doing something wrong, but rather in a different way.
- Perhaps maintainers need to take a lead in educating others.
- Doing PRs one-on-one can save a lot of time. Make use of both synchronous and planned review time.
- Keep an eye on team size and structure: sometimes the larger the team the possibility there is a problem.

# *How to manage a GitHub repo*

There were six eager and fresh participants in the first panel of the day. The key problem they discussed was: How do you manage repositories?

There was a consensus that handling repositories raises highly complex issues. They involve some key concerns:

> "Who manages it?"
>
> "Who maintains the project?"
>
> "How do you assign permissions to members of the project?"

One important issue that was raised was "organization". What does it mean? People were confused about whether the conversation was about the social act of organizing, as a verb, or the organization's ways of working, how they are handling GitHub. The combination of these various viewpoints is seen in the discussions below.

One problematic feature of a repository is that it is opaque. You cannot easily see or know who is managing and/or maintaining a project. There were discussions about having it in an open form instead.

Managerial issues were also discussed. There were competing views and conversations around wanting open source projects to be more professional, formal organizations, versus opening responsibility and ownership to everyone.

Arguments for making open source projects formal and professional were several.

There are projects with thousands of repositories that have professional teams that manage them. One of the maintainers shared the relief that such a set-up gave them: "it's nice to know that things won't break when you have dedicated professional teams to manage things".

For those who like professional maintainer teams, the general feeling is that GitHub's features are not really optimized for them. For instance, consistent naming is not optimized, and GitHub hasn't found a really good way to structure contribution processes or label them. They feel there are things that could be done better.

Participants suggested several arguments for having diffused leadership and open structures. One maintainer said they didn't want to have full responsibility for their project, so they manage the project by giving ownership and responsibility to everyone. Someone commented that this was a very anarchist way of managing the organization. Additionally, making people owners of your organization could be overwhelming in itself too.

A flexible structure allows people the space to walk away from projects. One of the key guiding principles is that if you stop having fun contributing to a project, maybe you should stop working on it. Participants generally shared the opinion that contributing to a project should be a positive experience, without guilt.

For example, one of the maintainers shared an experience in which they had a project open on the repository, disappeared for a while, and when they came back were extremely apologetic. It turned out that they had been involved in a serious car accident. The maintainer who was in the accident felt very strongly about how inadequate the car accident reason was. All of the participants agreed that the maintainer should not have felt guilty because they were not able to review a PR. One should be able to walk away and return freely.

Another idea shared was that a flexible structure is key to keeping your contributors. An open door policy allows people to remain in the loop, and also provides the possibility to pop in and out of projects if they feel burned out or feel a specific task is not for them. They should not feel guilty when leaving a particular project or moving on.

Barriers from Any Structural Decision Point. Whether one is in favour of pro or flex structure, someone raised the point that "social barriers could be removed by using similar ways of maintaining the project, but it wouldn't remove the other barriers". Any choice between a pro or flex structure could remove some of the barriers, but not others. For instance, uses of Github within private research groups or by maintainers as consumers of Github can solve certain barriers, but also present new ones.

Conflict. A common problem is the presence of conflict of interest and tension among maintainers of the same organization. There was a general consensus that there is a need for a system or policy that empowers members to kick out an individual, even if they were given permission by a particular group.

Empowerment. There is also a conversation about the context of empowerment in the submission and review of contributions.

Being granted permission did not make a maintainer automatically empowered enough to go ahead. They still felt the need for a project maintainer to review their work, even if the project maintainer was explicit about the changes.

One solution raised was that being more specific in the documentation could help people feel they are empowered and are project owners.

The conversation then turned to Github software as a form of organization. The group discussed UX difficulties. Some of the maintainers use GitHub Enterprise for internal issues and it really works for them. For example, one research project uses Enterprise with groups using the repos in a private space. As a research instrument, the repository, the maintainers, and the rest of the researchers act as consumers of the project, but they want to find a better way to have it be consumed other than GitHub.

There were participants who believe that maintainer profiles should be public.

There is a distinct advantage to having a maintainer team in a GitHub organization. It is powerful because then you can automate some of your work flows insofar as assigning PRs to the maintainer team. A lot of participants wished they knew that before. Some of the maintainers in the group were not using maintainer teams, so this information provided a very big moment of relief for some. Then there was a lot of discussion around terraform and using it with YAML, which would include things like the contribution files, discussing profiles, and a bigger space than a Read Me. Discussants believe that these should be in the setting of GitHub, as it helps the continuation of the projects.

With the context of a team, some participants discussed the posting of office hours, putting information in a place that is easily accessible, e.g. with the terraform, being more open with Read Mes, and having a maintainer file on how not only to contribute, but also describing the cadence of your project.

# Takeaways

- Github has an opaque repository structure. One proposal was to have it in an open form so one can easily see who manages and/or maintains the project.
- Maintainers' profiles should be public.
- Github settings should include contribution files, discussing profiles, and a bigger space than a Read Me to help with the continuation of the projects.
- Github should be optimized for professional maintainer teams: how to structure contribution processes and labelling them; how to have consistent naming.
- There must be a policy or system to kick people out, even if they were granted permission.
- One way to feel more empowered and have project ownership is to be more specific in the documentation.
- There should be an open door flexible policy in which people can pop in and out if they feel burned out or feel a specific task is not for them. They should not feel guilty when leaving a particular project, returning again to a project or moving on.
- Best practice: posting of office hours.
- Best practice: putting information in a place that is accessible.
- Best practice: being more open with ReadMes.
- Best practice: a maintainer file not just on how to contribute, but also describing the cadence of the project.

# *Licensing / relicensing*

There were around eight people in this session, including two students from Scotland and one maintainer of a large infosec project. Of all the people in the group, four were really engaged, the others were mostly listening. One participant questioned whether they were (themselves) actually a maintainer.

The discussion began with starting projects that are open source in the beginning, but with the possibility of monetizing them in the future (although the group agreed some projects were not meant to be monetized).

There was a lot of talk about licensing, especially around the Commons Clause. The conversation initially was focused by the question of the pros and cons of Commons Clauses, against other licenses such as GPL vs. GPL3.

The Commons Clause didn't seem to be well understood by the majority of the group. A simple explanation given was that it prevents companies such as Amazon from monetizing your open source project. Unlicensed was considered problematic in Europe and not well understood by the group (you can't give away your copyright).

A couple of people described it to everyone. They appeared to be the major contributors overall, with knowledge specialization in specific licenses such as CLA. To illustrate the Commons clause, one of the examples they gave was: if you are a large company with one legal person, almost any license that is not an MIT license is not going to be usable in your project, especially in Europe. They talked about how Google doesn't allow GPL, apart from under rare circumstances.

The group started talking about examples of Amazon monetizing open source projects in the past. They mentioned the Elastic search project. There was also lots of talk about GPL versus non-GPL, and how the MIT license was the best for most circumstances.

The group showed agreement on the interpretation of the commons clause, but they disagreed on CLA/pushing PRs to projects. There also appeared to be some disagreement and not much understanding about what CLEs were.

One of the participants felt very strongly about open source and not restricting use:

"I want people to use what I write; if they contribute back that's great. Not a huge fan of the commons clause. I'd rather more people use it than fragmenting the user base."

One person asked a question about licenses and restricting them to specific industries, or restricting specific industries from using them, particularly the military. However, the question was raised, "how do you define the military?" It was observed that this definitional issue could be a sticking point of how to enforce clauses:

> "The trick with clauses is that they're hard to enforce and they're more restrictive than you might think. E.g. how do you define military—any company that works with the military, contractor, government agency, school, etc."

In the group, there was confusion relating to the fact that there are some licenses that specify countries where licensing is defined/ defended. This was likened it to redlines:

> "For my project, we put it so that you don't sue us, but if you do…. Represented by a state I live in."

## Takeaways

- A good license to start out with if you perhaps wanted to monetize your open source project in the future is MIT as it is "just the more simple one"; you can put CLEN there and possibly change that in the future. Any non-MIT license is not doable in Europe.
- MIT is a license that's easy to start with for dual licensing. You could go with CLA and the possibility of a dual license. Force corporations to push changes back AGPL.
- A site that does a good job—which is free open source communities—is fosc.org. It talks about "business". For example, how does the license impact business vs. community?
- Sources mentioned to understand the different licenses included freeopensourcecommunities.org and choosealicense.com. For the second link, the group noted that there are more license descriptions than the website shows and yet more esoteric ones are included.

- JSLINT license means “you’re not allowed to use this project for evil”.

# *Diversity among OSS projects*

The discussion started off by mentioning Mozilla's biweekly diversity and inclusion call. The call is open to everyone and covers topics on how to be more inclusive and build a more diverse workplace. This involves, for example, talking about how to create a welcoming environment for first-time contributors or being mindful about the negative impact of a meritocratic culture. Examples mentioned were that white men can be overconfident in their skills or that experience requirements can be problematic because people judge themselves very differently.

Another example discussed was the [Firefox DevTools Slack] channel. It uses bots to help with translation or to give a brief introduction to new people. It also has dedicated channels to provide safe spaces, and new contributions are celebrated in the main Slack channel to raise awareness.

The group further discussed documentation and language as important factors in raising inclusiveness and fostering diversity.

Dr. Anita Sharma was mentioned. She studies contribution guidelines for OSS projects with an eye on how inclusive they are. Based on her research, she recommends making sure reviewers don't nitpick on grammar, since English isn't the first language for a lot of people, and instead either ignore small mistakes and to fix them later or hire an editor for documentation of heavy projects. It's also important to keep documentation language simple and easy to understand for non-native speakers.

Participants mentioned how helpful emojis can be in daily interactions and that it's important to avoid metaphors or things that don't translate across languages. Someone also mentioned automatic toxicity analysis to block comments from going online. On Discourse, for example, comments flagged as toxic go into a moderation queue.

Besides this, participants agreed that maintainers have the responsibility to monitor and step in when discussions get too heated, and to lead by example. In case of conflict, one option could be to escalate a problem to somebody else who is more experienced in dealing with conflict. Another option could be to do conflict training.

Other examples discussed were the Kubernetes project which has dedicated on-boarding sessions for underrepresented groups, Rails

girls summer of code and Outreachy. Another recommendation was Vic Health's, "Encountering resistance" which has a lot of good info on how to make change happen.

## Takeaways

- Be mindful that experience requirements in job descriptions can be problematic because people judge themselves differently.
- Be mindful of language:
  - Use simple language.
  - Don't nitpick on grammar.
  - Use emojis.
  - Avoid metaphors.
- Keep bikeshedding low, stay pragmatic.
- Create a welcoming environment, especially for new contributors.
- Create safe spaces for underrepresented groups.
- In case of conflict, escalate to someone with more experience if you feel uncomfortable.
- Lead by example.

# *Automating feedback to OSS contributors*

The main challenge considered in this session was that people don't read contributing guidelines, so they often submit changes that are not acceptable. When people make open source contributions they may not follow the contributing guidelines, yet they may take the feedback they receive personally. Human feedback can feel very confrontational, and it can make future contributions harder.

This session focused on the question:

> "Can we use automation to help prompt users to do this better?"

The group felt that people respond better when they get feedback on changes from a bot as opposed to from a human being. The example provided was that when a linter tells you to use a different style it isn't personal, but when a human tells you it can feel personal.

The group also observed that GitHub Checks do a very good job at enforcing policies because users feel uncomfortable merging changes when the checks fail. This approach appears to place the onus on the user to work out how to make their work conform to the guidelines and perhaps prompts them to read them prior to final submission.

People want to use bots for common tasks like linting, triage, tag assignments, stale closing, and validating changes, having sufficient code coverage and unit tests:

- Probot does a good job with a lot of problems.
- Stalebot is popular because users don't feel like it's as personal.
- Dependabot does a good job for dependency updates.
- GitHub Actions to bootstrap validations is great.

The group identified that there is still a need for more bots and automation. The biggest gap identified is the complexity of dealing with multiple repos in triage flows. They observed that Probot can't move issues between different repositories and that there was a need for more support for triaging with bots.

When asked, most participants said they had not contributed to the bots and are able to use them off the shelf. However, they appeared uncomfortable with changing their core code directly. When asked whether they were okay with giving a bot write permission, very few were. Those who did so mostly by creating human-reviewed Pull Requests (PRs).

The challenge identified by the group was that, unfortunately, bots can't do some kinds of validation easily, so humans have a role to play.

## Takeaways

- Bots help a lot because human feedback can feel very confrontational, and this can make future contributions harder. There are some good bots out there, but ProBot is one of the best for automating review tasks.
- CI tools shouldn't be the only validation tool that you use.
- It can be a bit unwelcoming if repos have too many checks. There's a balance.
- Most people have not contributed to the bots, and can use them off the shelf.
- There is a need for more support for triaging with bots.

# *Getting almost 100 people started with their first OSS contribution*

The inspiration for this session stemmed from a workshop at one participant's university. The workshop was called Open Core. Freshmen were mentored into understanding how to contribute to products. The goal was to train contributors in the basic skills of Git and GitHub. The ladder program initially starts with a template project, Bootstrap, in which issues are created with particular tasks.

There is no minimum number of PRs; instead, participants have specific tasks that are assigned to them. Each participant clones the repo and issues, and works through the issues one at a time, with different levels of difficulty. They are awarded points for how well they handled the issue. At the end of the program, participants will have the experience of contributing to a particular project. The session discussed how this program can be used to train contributors to support maintainers.

The following sections are divided into two parts: the program as contributor training, and how to elevate contributors into maintainers.

## Training contributors

The program was trying to solve the problem that most of the students don't understand development practices. This program is also an example of a special program to increase engagement with women who are minorities in their university.

The discussion then shifted to how contributors can become maintainers. Two questions were opened: Can this program be used in an open source maintenance setting? Can this program be used to get contributors and promote them into maintainers?

Some participants were not clear that the program would achieve these goals because the emphasis was on skill building and testing their skills. Two late participants were a little concerned that the program was a barrier to entry and would scare people off. As a counter, some argued that this might actually help people feel more comfortable

making contributions to the project: they feared many people were afraid to contribute because of the self-perception that they didn't have the right skills.

Other issues were discussed on how to encourage beginners. These include properly labelling issues and using automation to help set things up for new beginner contributors. One new attendee indicated that he has a project with 100,000 users, but he only has 3-4 regular contributors and no real maintainers besides himself.

The problem he is facing is that he has to do a large-scale refactor, but doing this means having a strong working knowledge of how the code works and being committed over time to make the changes, because you can't do this sporadically as the refactor goes on because you lose mental track of where you are. But most of the people involved are contributing only incrementally. They make a contribution, they come back in a month and make another contribution, but by that time things have changed and this creates frustration. To avoid this, he would rather give them a sense of being invested in the project, rather than being divested from it.

One participant suggested that awareness is very important to recruit contributors. He suggested that there should be an awesome list of projects that need help because frequently a lack of awareness that a project even exists is itself a barrier to getting contributions. Most people know about a project if they themselves are users. However, there are plenty of people out there who are looking to contribute regardless of the nature of the project.

## Challenges to becoming a maintainer

Another topic that was important was moving up into a maintainer position, besides having simply made lots of contributions to a project. Two people were involved in this second part, and it felt more like a lecture than a conversation about the problem. One said that frequently people put banners on a project if they are looking for contributors or maintainers to take over, but often those only go on too late, once burnout has already started, and nobody wants to inherit a dead project. In fact, there is often no way to contact the maintainer in that case, because burnt-out maintainers don't look at the issues. One suggestion was to have some sort of private messaging system to have a conversation about signaling your interest or other private issues.

Another asked if money would help contributors move into a maintainer position. However, the conversation there turned because money isn't going to get you contributors, although apparently Kafka does actually pay people to fix bugs that are on their list. They have a budget for hiring contractors, but this is not quite the same as having maintainers.

## Takeaways

- To train contributors: a university training program has one program to teach students how to "do" development practices; also used to target female minority participation.
- To train contributors: a program similar to the university, rather than become a barrier to entry, might be a pathway to break people's self-perception that they don't have enough skill to become contributors.
- To attract beginners: properly labelling issues and using automation to help set things up for new beginner contributors.
- To upgrade into a maintainer: a type of private messaging system must be in place to signal interest of another contributor/maintainer before maintainers drop out of site; traditional project banners means it is too late in the process.

# *CI / PR validation*

Many projects, regardless of technical stack or area of focus, are facing challenges around CI/CD. Centralizing on a single CI/CD stack has been tremendously helpful for various projects, and those that still have fragmented CI/CD and build systems struggle with scaling up.

Large scale projects, like Kubernetes and Node.js, effectively take a more liberal approach where the projects that fall under their umbrella can be more self-selecting about CI/CD systems. This doesn't necessarily mean the former statement about centralization on a single CI/CD system doesn't apply to these projects, but rather that the scope of their work has grown beyond a single focus and multiple teams are working on different projects and problems.

Several maintainers mentioned that they use Jenkins, but that it's perhaps not the ideal solution because of the complexity of maintenance. Nevertheless, people use it because it does its job well enough.

The group briefly discussed what specifically about setting up and maintaining CI/CD can be simple, and what can be challenging. They generally agreed that very basic checks are something that any CI/CD provider can provide, but running more complex checks is much more challenging to set up in any current CI/CD tool.

In both the discussion about Jenkins and the discussion comparing basic/complex checks, a similar point was raised: keeping CI/CD logic outside of the repos maintainers are working in is vital, and is a mistake that many of us have learned the hard way. It's an easy foot gun, and one that is perhaps not the most obvious to most maintainers who have not experienced it already.

The discussion then pivoted away from CI/CD systems and focused more on how maintainers work with their side of the process.

First, the group discussed how to address flaky tests–a problem that virtually every project seems to have, but which no CI/CD provider appears to provide any tooling or support for. Participants discussed the possibility of CI systems providing some kind of marker for maintainers to mark tests as flaky, but that it is incredibly challenging given that maintainers don't have direct input into those systems.

Additionally, the group discussed how maintainers can address and maintain flaky tests. There seemed to be consensus around having individuals on a team who can help address this directly, focusing on maintaining those flaky tests and fixing them, if at all possible. This also came up again later in this discussion when the group discussed "optional" tests, which could very much align with this feature. Having an additional, non-binary outcome for tests is something that projects at larger scales already have: Kubernetes is one example of this at scale.

There are many reasons why maintainers would like this kind of feature, including tests that may not be ready, tests for experimental features, tests that they want to use to collect information about specific code, and lowering the barrier to entry for new contributors.

Next, the group had a brief discussion about best practices around bad code–specifically, building out tests for known bugs so that maintainers can track that code over time, including regression tracking once the bugs have been properly patched.

Finally, the group discussed code coverage. Some, but not all, projects represented use code coverage. Most of the group agreed that in the context of PRs and CI/CD, failing a build on decreased code coverage is probably not something we want to focus on, but having an indicator of whether the coverage has gone up or down is valuable. That said, individual numbers over long-term intervals can be useful to understand better how our projects are evolving.

One of the participants asked what we thought the future of CI/CD was. There were a few different answers that the group found consensus on:

CI is super bespoke and is very personalized to each project. This probably can't/won't change, and the future needs to address this rather than forcing a mold.

It would be good to have a standardized pipeline that different CI providers can build upon. Effectively, make pipelines a commodity that everyone builds upon.

Supporting multiple operating systems is essential for the future. As our systems converge on xplat support, this becomes more and more important. At present, even supporting different distributions for the same platform is very important and not easy.

# Takeaways

- Centralizing on a CI/CD system is important to maintainability.
- Keeping CI/CD logic outside of core repos is important to maintainability.
- There's not enough tooling around flaky tests, and maintainers could all benefit from additional test criteria beyond pass/fail.
- Building out tests for bugs and edge cases is useful.
- Code Coverage isn't necessarily useful in the context of individual PRs, but it is more useful as a metric over time.

# *Reporting FOSS usage*

There were three people present in this session.

The participants noted that we may want to differentiate between companies that want metrics on their open source projects, versus independent/community projects. Companies may even want internal insights for their private repos/projects. So it's important to identify different kinds of entities on GitHub and understand their needs as well as maybe even label them as such: non-profit, university, company, etc.

Some metrics that are desired would be what versions of the software are being used (e.g. how many people are on the latest major version, who isn't upgraded, why not), smaller granularity (which functions/APIs are used or not so can make decisions or even deprecate them), where does a particular library fit within the ecosystem, or even where something is downloaded from (GitHub directly, package registry, separate site, proxy, forked repo, etc).

We may need to standardize a workflow to analyze usage (a set data format), and being able to reproduce results is a baseline, especially for scientific research. Example: each scientific paper should link to a corresponding GitHub repo.

GitHub already has data on programming language usage but is working on framework level data (e.g. for JavaScript, React/Vue/Angular/etc).

The new used-by feature may be a better signal than stars. It would be useful to have statistics on the licensing of dependencies, dependency graphs, and how this relates to funding and even project health.

Identifying/authenticating projects can be difficult. Projects can have similar names: which one is the official repo and which is a fork? Knowing the exact versions used may raise privacy/security issues.

# *Open source in closed organizations*

There are a lot of challenges when a company is engaged in either creating open source, or using open source software that is created by another company and managed by a company.

Companies are doing open source for a bunch of reasons, and may be "Open Source by name, but not by community". It makes it easier to hire people when people come in already knowing something about the product.

There is an incentive to open sourcing projects for PR, marketing, or to make hiring easier. If you open source something out of self-interest you may end up simply making the source code available. However, with only the source available there is a ton of missing context about previous and ongoing decisions.

When an open source project is maintained by a company and driven out of a company's internal resources without those internal resources being open it can be a huge challenge as the conversations about where the software is going and decisions being made are all happening on unavailable media (slack, email, meetings, etc.) at the company that has open sourced the software. Build infrastructure may also be a black box, with folks unable to see what has gone wrong, only that something isn't working.

What do you do if you are in a closed source environment but have open source components? What is it like if you are a maintainer on a project that is open source, but has private infrastructure (like React Native)?

For example, it can be a problem if you can't see the full CI/CD, and you know the checks are red, but don't know which tests are failing. Alternatively, what do you do if a lot of the conversation takes place on Slack, and the code is available, but the discussions and context are not actually available? How do you open this up? You can't just copy and paste it.

What if your source code is public by default, so people are watching you build in real-time, and the source is available, but the plan is not fully formed, or worse, the project is not a fully formed thought yet and may not continue?

For all conversations here, there is information lost or not made available. Historical / design information may not be not available; for example, if there are no design documents or architectural decision records in the repos.

## Takeaways

- Innersource is a good step to take for companies looking to open source in the future, to exercise the muscles in advance of actually open sourcing something.
- Clear signaling about how an open source project is being run is something that is tremendously valuable for those choosing to join an open source community.
- New European laws coming soon will require employers to report on the hours their employees work to confirm that employees are not working too long. This will hopefully result in better time tracking and more deliberate usage of time.

# *Issues with issues*

There were 12 participants in this session. Most projects have a lot of issues of varying kinds and have no idea how to get through the backlog. People leverage issues as a way to get help. There is a lot of automation around issue usage, and this creates a lot of visual noise. Many issues are invalid, but no one has the guts to say so and close the issue. Email notifications are broken. How do you keep the issues relevant and stay on top of them?

To begin, the participants went around the group stating the different topics that the participants maintained. The maintainers came from a place of having extremely large projects backed by large companies and large enterprises, to having small "hobby" projects, as well as ones that have many contributors. There were also some attendees from developer tool companies, namely GitHub.

A participant stated that they find it hard to get an overview or bigger picture. They don't know how many open issues they have. A participant asked the group, "Do you think that having too many issues is not a problem a maintainer should care about?" One maintainer who responded did not feel they should be concerned. Another maintainer says they do concern themselves with the number of open issues.

How do you deal with tech support issues? How many people are having the same issues? If it is one person then you can ignore it. But if you see it over and over again then you need to address it.

One participant said that they pasted the solution in the error message, but the users don't read the errors. There is an expectation that you should probably help people submit issues. GitHub has made it so easy to get access to maintainers, and has caused an explosion of issues being reported.

The participants talked quite a bit about solutions. One maintainer of a large project suggested to not use issues on GitHub but rather use them as email notifications. These could be filtered if they were issues created by the person, or if they were created by somebody else and mentioned that person, or a specific team.

A second maintainer who worked on a similarly large project associated with large enterprises mentioned that they cannot go a day without triaging or they become overwhelmed and burdened.

A third maintainer mentioned that they do a lot of automation and this is how they keep their sanity. A question was proposed of whether having a lot of issues was a problem that maintainers should care about. One maintainer answered that they have a lot of issues that they ignore because there are a lot of issues on technical support and they don't like to deal with those.

One maintainer who was not the original session host became the clear leader (they had an OSS Mac library project). They mentioned that they try to automate themselves out of jobs they don't want to do. When you first do open source you feel like you should be writing code, but when people start to contribute to your project you quickly end up doing mentorship, technical support or evangelism of your project. This maintainer suggested the idea of automating themselves out of these tedious tasks. At this point, others revealed other team structures in which individuals who like doing tech support or different roles will be elevated into doing those.

The group wished there was a role in open source where someone could only be a project manager. Overall there was a sense that nobody has much experience, and so all they can do is watch the problem. As a participant noted:

> "All that we complain about is how do we do the stuff we don't want to do and complain about how we can get better at the things we don't want to do."

Someone noted that this role does exist in open source projects that are managed by companies. Another person noted that there are lots of tools for project managers, but developers don't tend to know about them.

Towards the end of the conversation, there were a lot of examples given of project management tools. A lot of them discussed how problematic they were and how they worked for them. It seemed that most maintainers had not heard of these or the tools or did not actually want to use them. Bugzilla appears to be a product that no one appreciates; according to one participant, "Bugzilla is Jira with more AJAX". It provides self-hosted bug triaging. Nobody had an example of another issue management system. They also suggested it would be nice if GitHub provided the feature of automatically closing issues, or having pending issues rather than open issues.

# Takeaways

- Find ways for projects to have a dedicated project manager.
- Sort email list through Gmail exclusively to manage notifications.
- Another suggestion was made for a bot to tell you that an issue was stale and auto-close them and create more automation. This was also reiterated in relation to enterprise and non-enterprise cloud tools.
- Sometimes someone has a problem, and they need to ask for clarification. They ask a question and have to wait for the response. They would love for the issue notification to be in the queue.
- How to filter the issues based on the types in the CC address: set up some filters to specifically filter out non-important issues.
- If you don't know if you are making progress, try using ShipHub to create a burn-down chart of all of your issues.
- Develop a way to template questions with versions and reproducible bug templates.
- Labels issues with "needs more info".
- Rust compiler error message or A/B testing would be helpful.
- Add a secret message to add an emoji to get quicker triaging. React-native-firebase.
- Issues should be pending so that maintainers don't feel so bad when they close them.
- Managing notifications is challenging; It would be nice to have keyboard shortcuts.
- Octobox was mentioned as a solution to use outside of web notifications.

# *Community of advice*

There were three themes in this session: avoiding burnout, mentorship, and the relationship with technology that affects participants' ability to connect and grow. There was consensus among the participants that socialization with other humans gives you positive energy and can really recharge you to get back into maintainer tasks.

Online interactions can provide a false sense of connections, like through an avatar or a pull request, but you never really connect with the individuals as persons. More specifically, it is rare for maintainers to interact with each other personally outside of Maintainerati.

The conversation then shifted towards mentorship. You need to be careful in choosing mentors. One characteristic to look out for is someone who is providing feedback or advice versus someone who gives a criticism. It is really hard to find someone who really wants to help see you improve and see you do better. Everyone agreed that it is important to surround yourself with honest people, meaning people who will tell you how it is, even if they are not necessarily more experienced, who might have a different point of view or come from a different background.

Another characteristic to look for is shared values and motivations. Ultimately, of course, it is on you to choose the right mentor and even more so to choose the right answer. Mentors can give you advice and point you in a certain direction but it is ultimately your choice how you want to act.

One final piece of advice from one of the participants was to distinguish yourself. Your potential mentor might have multiple people who are asking for mentorships; you need to give them a reason to choose you over someone else.

One of the other participants noted that over time, as you become more comfortable and you grow, you can kind of become your own mentor and even have the opportunity to mentor others. In order to get there, it is really important to focus on self-awareness, being mindful, and being fully present.

Finally, the conversation shifted to discussing how technology affects our ability to connect, our ability to grow. One participant noted that

while it is a lot easier to learn technical lessons on your own, life lessons such as relationships, communications, and interpersonal relations are a lot harder to learn on your own and you need to rely on others to help with that.

Another participant noted that as a result of this, it is also important to disconnect, to reflect on that time online, but also spend some time offline. It is important to make true connections with other humans, real person to person connections, but this is hard to do again over a pull request or online.

## Takeaways

- Real, face-to-face human interactions are necessary to recharge and grant positive energy for maintainers who gain a false sense of human interaction online, such as through pull requests or interacting with avatars or fellow maintainers.
- The process of online disconnection is important to reflect on online interactions but also to establish true human connections and avoid burnout.
- Surround yourself with honest people who will tell you how it is, even if they are not are not necessarily more experienced, but who may have a different point of view or come from a different background.
- Mentorship is essential in human connection because life lessons like interpersonal relationships and communications require interaction with others.
- Choose a mentor who will provide feedback and advice, and champion your growth, versus someone who just gives criticism.
- Choose a mentor with similar values and motivations.
- A mentor or support network can give you advice but ultimately you have to make your own decision and pathway.
- Distinguish yourself so you can compete with other candidates who are eyeing the same specific mentor.

- The traits of being self-aware, mindful, and fully present can, over time, help develop oneself to become a mentor for others and yourself.

# *Dealing with "I want" and the lack of "I can" or "I will"*

This session was a small group that began the discussion with issues surrounding engaging people in projects beyond bug reports. The discussion built from the experience of an individual who, when taking over a project, found that a lot of people were very forthcoming in reporting bugs and issuing feature requests. They observed, however, that this did not translate to people contributing towards the project.

The key driving question for the group was how this dynamic could be changed: a movement from "I want" to "I can/will". This discussion also pointed to the tension point around making it easy to contribute to a project versus making it easy to be a contributor.

A hypothesis by the note-taker here was that the people who want are not people who can contribute because of a mismatched skill set. The y are not able to develop the said thing. It requires so much time and investment to learn how to build/develop this.

The group discussed the experience of a particular participant. The participant's current approach is that when people submit requests asking for things they say:

> "Sorry, I can't work on that now but I would be more than happy to see a pull request if you want to submit it."

Another participant asked them:

> "Why do you feel the need to field these people's pull requests? Why do you feel the need to respond to issues?"

The first participant didn't really have a clear answer, other than that he wanted to. The two participants discussed reasons why they weren't getting pull requests. It became apparent that the participant had taken over his project about a year ago. The question was asked, "How did you get suckered into this?" Their response was that they had been looking for a project to contribute to within the open source community and that just happened to be one that he used and felt that he could take over.

Finding something that interests you may not always be the sole criteria to use when searching for a project to take on. The issue here, consequently, may be about identifying what a sustainable project looks like, if you are going to take one on. The discussion and experience above may be a useful guide to those looking to take on projects about what to look for. For example, when one has a popular project that doesn't have a lot of contributors, maybe it's just not sustainable.

Getting people to follow through also appeared to be an issue. However this may be more to do with the language used, which in this case is Go. Not that many people develop in Go and there is a corresponding small number who want to learn it. This was suggested as one of the reasons why they might not be getting so many contributors.

The person who maintained the Angular project said they get a lot of people who want to contribute, but then they find that it might take them a few days to get up to speed and fix random bugs. Even with contributors, it still appears that a lot of people become disinterested and lose the motivation to actually follow through:

> "We get a fair number of people who want to contribute. It'll take a good few days to teach a random person. Even bug fixes are fairly complex and require restructuring stuff. You need to be extremely motivated to do any of this. Only self-motivated people do it."

When the one participant found out that the other participant was on the Angular tools team, he became very interested and asked lots of questions. He is a huge user of Angular and considers himself part of the community. They talked about ways to contribute to the project. This is more so for the Angular project than for the first participant's projects, who asked: "Angular has a huge ecosystem, have you ever thought about looking into this?"

They also talked about how easy it is to find contributors to code, but they are always looking for good UX people and documentation people:

> "Easier said than done. No incentive for them. No recognition. Not on GitHub. Not on Slack."

There is no incentive for these kinds of people to join the project as they don't get recognition. They are not on GitHub. Participants don't have any answers for this.

The group also noted that awareness of the maintainers' community is not strong among participants.

## Takeaways

- Create a road map for your vision, and clearly communicate what you're going to be working on.
- Select projects to take on a function in ways that meet your expectations/values, for example being part of an existing ecosystem you are active in or being community-oriented.
- We need matchmaking for open source, finding contributors in documentation and UX beyond GitHub. Maybe ServerFault is better, or contributors may be found at Meetups. It is like "online dating versus spending time with like-minded folks".
- Increase the visibility of the maintainers' community.

# *Funding*

More than 25 people turned up to discuss issues of funding, with note-takers indicating that maintainers are very concerned about funding. There was some policing among the participants in terms of communication (terms of speech or voice) surrounding this topic, also suggesting that it is a "hot button" issue.

Maintainers feel like asking for funding feels like asking for charity, but nobody can really make a living off open-source even though their work provides huge value to corporations. There was a strong feeling that maintainers are enriching corporations and getting nothing in return, and frustration around companies making profits from what basically amounts to free labor. One note-taker observed that this session focused on a question of platform work and the platforming of that work.

Most issues discussed by the group centered on getting funding from corporations as opposed to individuals. The group identified that companies have money for open source software, and asked how they are helping with funding. They suggested that mostly they aren't. Companies may be providing additional marketing support for a project, "but that doesn't pay bills". The discussion was about open source's sustainability, and the tone of the discussion suggested frustration regarding the options available.

Participants discussed sponsorship programs and many users said that it felt like a charity program:

> "Sponsorship doesn't feel right because it feels like charity. We're doing real work. Pay us for the product we are delivering!"

"Moving out of the charity model" was identified as a desire by participants, implying some consensus about the problematics of charity. For one participant, visibility is a key driver for the charitable approaches of corporations. For the group, GitHub Sponsors sounded cool, but some people don't think they will be able to bring in enough money to make a living and that it still feels like charity. A lot of people in the room felt that we need to be moving away from a charity framework.

Participants discussed the question of how to receive (charity) payments from a tax perspective. They generally felt that if they were only getting a couple hundred dollars a year, the tax overhead was not worth it. How do you receive (charity) payments, for example, and explain it to the German tax office? Consequently, receiving donations was perceived to be "a hassle" because you need an entity for it, no considerable money is moving that way, and there is a lot of overhead.

Another financial general challenge identified was that corporations sometimes need contracts to contribute large blocks of money, which makes contributing to open source more complicated. This is harder to work with as an individual.

Evidently in the group there was frustration around the kind of patreon and grant/ sponsors model for participants. In weighing whether grants feel better than sponsors, and whether they achieved more, participants suggested that it depends on the expectations of the grant. They observed that if grants can send money to people and organizations, it can be very impactful. One person observed that they want to keep parts of the stack from "the big five" so had established a grant scheme, and were accepting applications from a diversity of people and organizations, both small and large.

An underlying topic throughout was who has the ability to invoice, which requires entities (Patreon and PayPal were mentioned as examples). A participant had done some random sampling on Patreon, seeing who is sponsoring whom with the main sponsors being other developers rather than corporations. The group observed that:

> "I don't want money from other contributors! They gave me time. I want businesses to pay."

Participants suggested that sponsorship feels the community is paying the community and just moving money back and forth. They estimated that 80-90% of funds never leave the system because maintainers sponsor each other. This does not really solve the problem fundamentally. While maintainers might sponsor each other it's a zero sum game if outside money isn't part of the funding model.

A critique offered was there is "slush money" being passed around, 80-90 percent of the flows go from pocket to pocket, and the pushers skim a slice off from each transaction. In this situation only some individuals may make a lot of money. One participant felt that this is a harmful mechanism, and asked, "how do we break out of this?"

Another participant insisted that framing of sponsorship is at the heart of the issue. The current framing is "the software is free, but I am awesome". The observed that this enacts a perverse mechanism where it is not aligned, whether the logic is "the value I get that I pay for", or "I pay because I appreciate that there is a person behind it".

Participants discussed the question of when you do set up some sort of funding for a project, how do you decide what kind of value corporations are looking for. Participants discussed what sponsors could get for their support:

- A seat on the steering committee.
- Consultancy.
- Maintenance (which was seen crucial).
- Buy feature development (like Kickstarter).
- Create a feature faster.

In terms of offering a seat on the steering committee, participants observed that while companies may pay an enormous amount of money for a seat and set the direction, maybe that wasn't creating the best outcomes for the projects.

Consulting is another option discussed. An issue raised was that while maintainers can consult with companies that want to use their software, they don't get to push the work they do for consulting to the open source projects in the first place. Consultancy and maintenance are juxtaposed, and participants highlighted the difference that the consulting model may often only support for active development.

Participants felt that consultancy takes away time from maintenance and development, which was seen as critical: "By maintenance you gain knowledge by theoretically having thought about those issues, and you can then do consultancy" to get the money. A note-taker asked, "Is maintenance in this respect almost as career development?"

Maintainers are willing to explore alternative funding models, including brands, bounties, dual licensing, or even selling their products in a SaaS model, but all have unique challenges.

Participants considered Kickstarter for features and observed that you get paid to do a feature initially, but long term supportability is not

funded. With the RedHat model the software is open-source, but users need to pay support contracts. For dual licensing, they observed that it was free for open-source, but commercial needs to pay. The SaaS model involved selling the product as an API or service instead of offering the source for commercial products.

Bounty systems/mechanisms were described by one note-taker: "basically they want to hire you, though that's not what it is called, and implementations go through their approval for you to get paid for." Money, sponsorships and bounties were also considered to be a prioritizing ranking/mechanism: "here is my issue list that I am working on anyway, I want to make some of these faster if I get paid for them."

Participants observed that bounties still feel like there's too little money for it to be worthwhile and that bounty sources are not profitable enough, having tax complications. A joke was made about bug bounties: "get paid once, maintain forever". An example of one way to approach this provided was that a suggestion had been made by one participant within a platform work product to pool a maintenance slice off each of the sponsorships/bounties.

## Takeaways

- Industry-maintainer collaboration surrounding solving funding problems and challenges. Maintainers continue to want to see more leadership out of the industry on how we solve these problems together.
- Linux Foundation sponsorships are sufficiently large enough that they can actually help pay the bills ($50k minimum).
- How do we fund projects instead of people? Projects need a tax entity and a payment structure to redistribute funds to the individuals who work on it.
- A participant recommended to use https://backyourstack.com/ but raised the questions on how to dispatch the money: equally, or is a dependency more or less important?

# *Non-technical aspects of OSS versus career growth*

This discussion was between two participants, plus the note-taker, who also participated. It felt more like a therapy session than a real conversation. The conversation began with discussion about a Twitter comment on how women shy away from using titles that are seen as less serious and less technical, such as developer relations or community managers, and how often women and minorities take over the DNI work because they are the only ones who actually do it in the first place.

## De-valuation of glue work

Glue work is that necessary work of coordinating people, especially across different functions, and ensuring that everything is moving smoothly. Glue work is rarely seen as valuable, precisely because it is not seen as additional work or even actual work, despite its importance in the smooth functioning of an organization. Thus, it never results in a sense of increased responsibility. People who do a lot of glue work are passed over for promotions and raises because they didn't do as much engineering work. This disproportionately affects females. Men often don't need mentoring or don't think about mentoring because white men are more frequently given opportunities that women and minorities, who need to beg for them and are often denied. We need better leadership from CIS white men, especially when they are in a better leadership position.

Glue work is real work, especially working in a cross-vendor effort which compounds the problems that arise. Despite good leadership, you still have to deal with a potential lack of inclusiveness from the other vendors. The result is you have conflicting policies and conflicting ideologies, and you have to find ways to work past that. This is under-appreciated, difficult and tiring work.

An example was given by a female participant who works in an open source cross-vendor project. She wanted to spend more time in engineering work, but there is so much up front work such as setting up the infrastructure and instrumentation, and other tedious tasks. As an OSS community, the problems are larger. You have to set up the

community in order to be successful and inclusive. She doesn't want to do that work; it's not really engineering work, it's work that everybody says is important but nobody really acknowledges it and nobody wants to do it because it's not the fun engineering work.

Since you can't get funds to hire somebody to do that sort of work for you, it becomes "natural" to fall on her as the woman who notices that the work needs to be done. But it is difficult to get buy-in from management because they take an attitude that it's not valuable to the business. The implications are that the community comes second, front work doesn't get done and ground work is messed up.

The devaluing of glue work results in lowering cross-project contributions. One participant voiced that he has a low tolerance for making random contributions to other projects, because there is a risk of getting into situations with a different culture and non-inclusive environment. It is not that he doesn't want to contribute to a project, but he doesn't want to deal with bullshit after you make a contribution into a project just to fix a bug or a typo, or something like that. There was more conversation about the cultural clash and working on OSS in a cross-vendor way.

The more common solution is to revert to closed source development because the models of contribution are much clearer and they don't have to take the community into account. For this reason, a participant commented that it becomes a legitimate reason for a company to hire a developer relations person, so that they can work in an open source environment. The specialist developer relations person can handle the community aspect of the open source stuff while allowing the other engineering employees to focus on the engineering work itself.

## Strategies to Re-Value Glue Work

Several participants proposed to change promotion metrics or inclusive technical contributions. The point was raised of how you measure people's contributions. One participant mentioned that, in his company, they also looked beyond technical contributions and included mentorship and industry impact as criteria. This can help offset some glue work issues.

Another proposed solution is to decouple the review cycle from the promotion cycle, as this hampers career growth. The primary casualty are the women:

> "It's always useful to ask how many women there are in staff engineering roles, because frequently the answer is going to be 'not nearly enough'."

One reason for this is that review cycles don't come often enough, and so feedback comes too late to change your behaviour. The result is that you failed to gain the promotion because of the lack of timeliness of the feedback.

Finally, the group discovered that many of them had backgrounds in the humanities. They all agreed that the world needs more humanities in tech to overcome the bubble that prevents people from addressing the actual issues that they need to deal with. This included the thought that "nobody seems to know what they are doing, but they certainly think they know what they are doing, especially in the San Francisco type bubble."

## Takeaways

- Glue work is undervalued and affects promotion metrics and morale, especially for female developers.
- Re-valuing glue work could benefit from better leadership from CIS white men, especially if they are in better positions. One impact of good leadership is an increase of cross-project contributions.
- Closed source development is one solution because the models of contributions are clearer and the community is not a factor at this stage.
- A closed source development model of contribution legitimizes a need for management to hire a specialist developer person to work on the OSS environment.
- Change performance metrics to re-value glue work.
- Propose inclusive technical contributions.
- Decouple review cycle with promotion cycle due to the uneven timing of feedback and change possibilities that disproportionately affects women.
- Humanities in tech background helps in overcoming tech silos.

# *GitHub notifications are terrible*

There were six people in attendance, some of whom are designers.

The problem discussed is that not every notification you get is important/contextual, so we need the ability to filter them out when necessary. It may be useful to try different workflows in different situations/teams.

When you only care about a single repo, pinging people by name works well. But notifications aren't scalable for big teams: you end up with multiple organizations and repos and unable to track everything down. Over time the core people get busier and you end up with teams instead of a point person and the responsibility is diffused.

Part of the issue is just identifying the right people to work on tasks (who cares/owns/understands what, what is the scope). Maintainers end up on third party software to manage and even communicate with people (direct message on Slack, Twitter, etc) since you know they won't see the notification. People feel guilty for missing something, but in the end declare notification bankruptcy because it is overwhelming and one might as well start over.

Every comment creates a notification so you need filters to get rid of messages from CI/bots. Some people get good at creating advanced github queries (only show things that were commented on in the last week). Projects have their own custom dashboards with smart filtering of the project's workflows.

On stale bot: people have opinions on how annoying/useful it can be. Some don't use email since they don't want to mix notifications in with other mail.

## Observations

- Notifications are so frequent that they can easily drive your life and give you a false sense of accomplishment. How much time did one spend on GitHub?
- Using emojis to track and comment is fun.

- Maybe notifications isn't the best name since most people don't actually want a push notification. It's more of a task queue than something to immediately answer to.
- Dead issues can sometimes blow up into huge threads if they aren't locked.
- Simply triaging may signal to others you want to continue to engage when you don't.

## Takeaways

- Try out Gitspeak or Octobox. Simply unwatch repos.
- GitHub has a new feature to only be notified for custom events (when an issue is closed/reopened) versus all the comments.
- Unsubscribe from threads often.
- Take advantage of the new triage role from GitHub that helps new contributors share the load.
- Turn off the default setting of watching all new repos to an organization.
- Use GitHub teams versus direct collaborators so you can audit who should/shouldn't have access to the organization.
- It can be useful to set aside specific time for issues/triage, batching it up, maybe in the beginning/end of the day. For other people it's better to just do it when it happens.
- Mari Kondo your issues, just unsubscribe.
- Create a tier of reviewers, like having a +1 review for formatting/basics and a +2 review for how it fits within the whole of the project.
- Encourage volunteers to review other's PRs rather than just their own.

# *Contributors*

Projects that spend a lot of time focusing on onboarding contributors have an easier time (not an easy time, just easier) onboarding contributors. The best practices we have seen include a dedicated onboarding path that has a dedicated sub-project, with labels for "good first issue", and special interest groups so that as folks come onboard they can self-select into areas of expertise. These dedicated paths are probably not feasible for projects with only tens of contributors; however perhaps groups of maintainers with a similar focus could get together to create an onboarding path.

An "open source brand guide" would be super useful as a starting point for folks who need guidance on how to communicate with everyone. This would help with onboarding new contributors. For example, if you joined a company you would get a brand guide to tell you how to talk to customers and communicate externally on message, and in the proper tone. Open source folks lack this tool.

Communication is hard. In a perfect world there would be time for a 30 minute Skype chat to walk everyone through their first pull request.There would be established trust, and empathy would exist between everyone. There isn't always time for this, and the shorter / terser communication becomes, the more cultural barriers and other factors impede communication.

This makes it more likely that a miscommunication will occur or that someone will not feel "heard", or will feel "insulted" by someone else. This is an independent challenge for onboarding new contributors and making sure that they are given the benefit of the doubt, and that they give others the benefit of the doubt during communication.

Some projects need non-code contributors, and it is not widely known that you can become a contributor to help with documentation, triaging, or other non coding tasks.

## Takeaways

- Setting clear expectations for new contributors in both directions is key. The contributor needs to tell the maintainer what they hope to get out of it, and how long they are interested, and the maintainer should tell the contributor what help is useful.

- Localized onboarding guides are definitely helpful.
- “Guidelines” aka - terms of conduct - would be very very helpful.
- Someone’s first contribution should be handled specially.
- Letting people self-select into their first issues has worked well to develop longer term maintainers. These are sometimes managed through an onboarding sub-project to provide additional guidance, and even fill with some of the easier first issues.
- There is a label for “good first issue” that means you are willing to help people fix the issue, and guide them through the process. You can volunteer to help folks fix this issue.
- There is a label for “help wanted”. This means the filer can’t really help with it.
- There are special Interest groups (sometimes up to 40) within a project to let folks self select into an area that they want to contribute to the most.
- Leave typos in the “onboarding” guide within the onboarding sub-project. See who fixes typos.
- Keep in mind that folks who don’t speak English well may still be able to program in English.
- Documents should have a “contributing.md” and a “how to review pull requests.md”
- Take care to rephrase things so as not to be rude; for example, “Yes, and” instead of “no, but”.
- Take a look at the GitHub Repos “Maintainers Wanted” and “Danger”.
- Dealing with hostility to new contributors: if maintainers are consistently assholes, they get removed.

# *Making contributions more open*

As introductions were being made, the participants explained how they started their open source projects and how they lacked structure. That same lack of structure was the first theme. Attendees gave examples of how they got to this point with no organizational structure or written documentation.

The beauty of open source is that you can start without permission or a plan. But then, participants asked:

> "How do you start developing in a serious way?"
>
> "How do you go from a project that is a few people to a place that has accountability?"
>
> "How do you get maintainers to be more accountable?"

The problem can be that a project can grow bigger faster before governance is created. They get stuck in a situation where everything is only tribal knowledge. There were mentions of how open source always starts this way, and how you will eventually need to make time to organize thoughts and contributions in a way that all current/future contributors can participate.

Much of the discussion focused on the case of one project that has poor maintainer accountability. The engineer who works on this project explained that when they start work in the morning and the project is broken, they always have to fix it because they are the only one who cares. They have light review guidelines, but they are often ignored. A participant commented that this is a culture problem: if maintainers don't respect the rules, then they should not be part of the community.

A participant offered a second example of a large academic project that works in the open and struggles with consistency and communication due to the project having multiple contributors and no clear technical leader. At this point, the open source maintainers started answering specific questions for this project based on their experience in a non-academia realm. From this point until the end of the session, the participants shared examples with each other and the conversation was very supportive.

## Takeaways

- You need a governance system at the beginning.
- If you join a project later, you have the opportunity to provide more discipline to the project: this is time to discover what information is not written down and needs to be.
- However, you cannot write everything down.
- Some contributors need to be more attentive to preventing code from breaking things.
- You can make a video call if synchronous communication is needed.
- Codes of Conduct can be version controlled in Git.
- Contribution guidelines are essential, but they are only effective if they are clear.

# *GitHub package registry*

There were four attendees and three were from GitHub Product or Packages. The group turned more into a panel with the engineer participant asking questions from the panel. There were discussions around the following issues:

Questions specific to package registry, for example that the documentation doesn't explain the limits very well.

Conversations were discussed about security and security issues. For example, a maintainer was concerned with the layers and also asked if there would be support for commits too, so the engineer was able to ask a couple of those questions and being able to assign with the GitHub API and ask questions about how the product was actually built, e.g. in a silo fashion by layers.

Finally, the discussion turned to the future plan to open source the back of the package registry so you can create your own support. There were conversations about using it for different projects, security and trust, more specific to the maintainers' full-time job, not really related to the open source activities. This mostly was a product discussion and concerns were based on their day-to-day work, which included security concerns and documentation needs.

## Takeaways

- Clarify the package registry's documentation to explain the limits, security layers and how to create your own support to effectively make maintainers' jobs easier.

# *Life of an OSS contributor + working from home*

There were eight participants in this panel, and they were completely engaged in talking about the challenges of working from home. A natural progression of topics were covered, beginning with the challenge between ways to focus on work while at home and the ways they were distracted from it.

Focusing on a project appeared to be a double edged sword, in which getting to the point of focus appeared to take a wide range of tools and approaches, while stepping away or switching off from a project was not always possible and bled into family time:

> "You want to be present for your family but you know, and they know, you are thinking about 101 different problems in your head at the same time."

Social media was found to be an unwelcome source of distraction, with some participants of the group limiting their time on it and others completely blocking it. All participants discussed transference of distraction to some other form of media as a substitute rather than becoming successful at focusing.

The line between focus and distraction was additionally blurred by the ways family interrupted because participants were perceived to be "at home".

> "The perception is from your family that if you are working from home you're interruptible, but you're not always that flexible in reality."

All participants found that the flexibility of working from home is both a pro and a con. Because of this, they felt that their work-life balance got mixed:

> "If you work from home you're available from 9am until 9pm."

Other key challenges to working from home mentioned is the possible reduction of social interactions and also not having a colleague to ask random questions. Similarly, even when you are not working at home, you are still in the context of "work".

The presence-absence and work-life tensions that these conversations point to appears to be created through two dynamics: being passionate to be working on OSS projects (it didn't always feel like work), not holding to a demarcated time and place to work on these projects, and trading away time with family to the open source project because participants wanted to or because they felt that people were relying on them.

## Takeaways

- Participants attempted to manage availability to family and other responsibilities both spatially and through mind set, including assigning a specific space for OSS project work and making a conscious decision to switch to "family mode" when at home.
- Switching context (ie going to a coffee shop) and using a structure may be good for some people as a way to contain the times that they work in their OSS projects.
- Tools and preferences that people had for working remotely included a website called focus.me and using noise-cancelling headphones.
- Utilize procedures for "switching moods" such as background music to create a set and setting for the work.
- Focus may be assisted by stepping away briefly to return fresh to the work, for example taking a 20 minute break every hour, using mediation tools or napping.
- Defining the OSS project as work makes sense, even if it's your free time, or something you do for fun.
- Limit yourself and passion for your OSS projects.

# *Being a good OSS citizen*

"Do you know how to onboard?" was the introductory question for around 20 participants in this session.

Some projects have a dedicated on-boarding path for new contributors, which often starts with fixing typos or other easy-to-solve issues. Participants discussed what a "meaningful contribution" is and whether updating the documentation or fixing typos falls under that. There is a special sense of ownership and pride in regard to code contributions. So these are probably the most significant contributions, since maintaining the code is the point, but also a dissenting voice:

> "As maintainers we all looooove documentation and project maintenance contributions. How do we celebrate this?"

The group talked about the GitHub contribution user interface where reactions or comments don't count in the "Contributors" stats. This mirrors the previous discussion point that only code contributions count, but it seems that GitHub is going to address this issue by updating the rules on what counts as a contribution.

There is still tension in having "toxic and unhelpful" mechanisms like leaderboards that focus on huge contributions and might discourage new contributors who might think "I will never ever make it to this list". The group wondered what a good contribution list could look like and if it would be better to have a monthly contribution list or a flat one, like what WordPress has chosen.

A lot of projects also use "Good first issue" as a label or have a "Maintainers wanted" call to indicate entry points. Another helpful tool is a CONTRIBUTING.md file translated in languages where core or future communities of the project are.

The discussion moved from onboarding to the review process and participants remarked that a review process heavily depends on the reviewer and is in general an inconsistent experience for contributors. Participants discussed further that it might be helpful to review the review process and its guidelines.

One participant recommended to run exercises that help to reveal and reflect on hidden biases to prepare reviewers better. Another suggestion was to outsource some of the process to tools, such as linting

or tests to deliver a more consistent review experience. Also, contributors can run it before handing the code in for review.

Participants agree that it is highly recommended to be explicit and transparent about the contribution and review process and guidelines, for example by having a Code of Conduct (CoC).

Asked about their experience, two recent computer science graduates said that they rely on the README file and agree that the first experience matters a lot.

Local hackathons or regular calls can be a good way to onboard new people and add a human touch to the contribution process. Another helpful thing might be having workshops for new contributors and video record these to make them available to a broader audience, and to have a dedicated onboarding team.

## Takeaways

- Design an onboarding path for new contributors and document it.
- Translate process/on-boarding guidelines to languages where core part of the community is.
- Use clear issue label to indicate good entry points for new contributors, for example “good first issue”.
- Design the review process and document it.
- Use meaningful tools to help improve the review process (tests, linting).
- Make sure reviewers follow a Code of Conduct, are open to new contributors and sensitive about their own biases.
- Consider documenting the on-boarding or review process via video or to run dedicated on-boarding events for new contributors.

# *Cross-discipline contributors*

In this session participants discussed the challenges in engaging with non-traditional contributors such as UX designers, illustrators and document writers. They found that this problem space had a few areas that need attention including making projects discoverable, supporting non-code contributions and recognizing these and driving sustained engagement of non-coders across projects.

Discoverability: how do users discover opportunities? How do we reach non-traditional contributors in the first place? How do we influence the tech training programs and meetups to help engage them with open-source?

Making contributions: how do we make it easier for new contributors to make changes? It can be tough to find people who want to work on documentation/editing in the open-source community. Tools like GitHub can have a steep learning curve for non-technical users. Should GitHub be easier for non-technical users? Not be so married to git terminology?

Recognizing contributions: how do we help provide meaningful recognition of non-code contributions that help users feel good about what they did, and provide them some tangible value?

- It's a mix of culture and tooling.
- Can we make best practice videos on how to do this?
- Can we help roll-up likes to help establish a user's recognition? Achievements might be good.
- YouTube plaques for vlogger milestones are a great way to celebrate users.

Driving sustained engagement: How do we help a non-traditional user move from one project to the next. People get engaged in OSS from work, but it's tricky if their communities don't engage there. People who aren't programmers don't know about GitHub, so they may not be aware of the opportunities. How do we reach these people? Can community outreach help? There's lots of "learn to code" outreach, but there's very little to help users discover the other roles in tech.

Participants also discussed the issue of contribution by academics. They asked, "how do we influence academia to contribute to open source instead of doing greenfield project?" This suggests that whilst engaging academic contributors is desirable, academics are more likely to have a different agenda to the broader community.

## Takeaways

- The challenges involved in engaging non-traditional contributors include making relevant projects discoverable, facilitating contributions, recognizing non-code contributions and driving sustained engagement.
- Platforms like GitHub can do more to make their project's needs known by putting up global tags like these and making them super discoverable. For example: "Good first issue"; "Documentation needed"; "UX design needed"; "Icon designer needed".
- WriteTheDocs is a great conference that supports cross-disciplinary engagement in non-code contributions.
- Recognition of non-code contributions involves a mix of culture change and tooling. Suggestions for tooling include: best practice videos on how to do this; establishing a user's recognition through badging achievements; provision of YouTube plaques for vlogger milestones.
- Helping a non-traditional user move from one project to the next drives their sustained engagement.

# *Power of positive feedback*

The discussion consisted of three participants, who shared toxic issues. These were mostly around reflections and learnings on positive and negative feedback. Some key issues include:

> "What is the role of positive and negative feedback?"
>
> "How can we use these types of feedback judiciously?"

Context: One participant pointed out that it was horrible to wake up to negativity in your GitHub feed. The host of the session said that he "sometimes opens an issue to leave praise and thanks, and sometimes the response is "What the fuck is this? Close.", but mostly he gets sincere appreciation.

One insight that is that human behaviour is "really good at pointing out what's wrong—just look at the name "issues" and "Git blame"—but we're frequently not very good at pointing out what's going right. Why is this?"

The group came to the realisation that rewards reinforce the behaviour and culture that you want to build. It's not just about recognition. It's about positive reinforcement. The group also talked about not wanting to create praise junkies, that is, people who just do things to receive praise.

The host then began to wonder if, as he opened up these issues, that was what he was doing, whether he just wanted to receive the praise and gratitude for opening those, because he was certainly not trying to reinforce a particular behaviour. He was just trying to be nice.

This is where the conversation started to take a bit of an interesting turn away from waking up to negative issues to asking, "What is the role of positive praise?"

## Dangers and Counterpoints to Positive Feedback

The group talked about celebrating achievements versus cults of personality. The group talked a bit about work experiences where reward systems quickly turned into cults of personality where you have

to be a member of the in-team or cool kids' club in order to be eligible for those kinds of rewards.

Then the group thought about people who use praise to manipulate others. There are certain sorts of Machiavellian people who are very good at getting what they want out of people by using honey rather than vinegar. This is because "it's nice to be told by strangers that you're valuable." Slowly a person begins to tailor issues to garner praise.

One participant worried whether his frequent entertaining talks without substance "was just performance and if he was just doing it for the praise." However, there is a role for this. For example one member pointed out that even if tech conferences are "tax-deductible hangovers" or that you don't learn from it (with some disagreements here), "you make friends and connections with attendees" and "also get inspired to do things" and "and that's what it's (conference) for."

This is when the host realised that crafting entertainment is maybe alright, because he wants people to tell him how good it was, so is he thinking of himself more than the audience? A counterpoint to this was that part of the purpose of the talk is to generate conversation and to meet people:

> "Thanking you for your talk is a great excuse for somebody to introduce themselves to the speaker, and that frequently creates a positive environment for collaboration in the future. So, it's not all bad."

## Advantages to negative feedback

Engineering control systems need negative feedback. One participant mentioned that positive feedback creates unstable systems in engineering control systems. Some examples include the PID controllers or the steam engine governor which are both exclusively regulated by negative feedback.

Negative feedback systems are designed to say, "No, that's wrong, go back to what is right" when these systems move away from the desired norm: "Negative feedback is designed to subtract behaviour." Meanwhile, positive feedback can only create unstable systems by saying, "Great, keep doing what you're doing, go faster." Eventually,

"the system will self-destruct or reach the speed of light. It's not typically what you want."

The reason that it is easier to give negative feedback is that "it has a clear action item associated with it. It does have an emotional component and it does set up incentives for certain behaviours…" In contrast, "positive feedback has no action item associated with it in the same way."

The session host commented that they originally gave positive feedback to make people happy, but now realizes that it can be used in a more structured, careful way to reinforce future behaviour.

## Takeaways

- Positive feedback feels better but has a tendency to create "praise junkies," cults of personalities, and the manipulation of others. Therefore it needs to be used more carefully and structured to reinforce future behavior.
- Positive feedback during a tech conference creates a positive environment and rapport with speakers and allows for interaction and creating of future collaborations.
- Negative feedback is essential in the system because it provides an action item that sets up incentives for certain behaviours, unlike positive feedback.

# *Introducing funding into community-driven projects*

The group agreed that they were going to intentionally avoid the "how to get funding" topic and instead discuss "how to manage funding".

One of the first examples given of how to effectively manage funding was the Node.js Travel Fund. This is an amount of money that the Node.js project requests from the OpenJS Foundation (formerly the Node.js Foundation) every year to enable its collaborators to travel on behalf of the project. All of this is logged publicly and managed via PRs inside of a repo. The only part that is not fully transparent is the submission of receipts to the Foundation so individuals can be directly refunded. OWASP has a similar process, except for requests, which go into Jira.

Stepping away from specific examples, everyone agreed that having clear and well-defined rules is very important. Defining what the project considers to be acceptable and unacceptable uses of funding is vital for the long-term success and sustainability of funding models.

OpenCollective has laid out a foundation for this, effectively allowing people to request funds and having that approved by core collaborators, but they may not have the tooling in place to explicitly define acceptable/unacceptable usage. Additionally, the model of OpenCollective, allowing core maintainers to manage funds, may not scale when there are dozens or hundreds of core maintainers.

One group member brought up a rather interesting case: Debian. Debian keep money in different organizations around the world. In taking this approach, they avoid the need to convert one currency to another and incur costs for exchanging currencies. Additionally, they have historically not paid for maintainer time. This is beginning to change, however.

Something that no direct funding tooling currently helps with is budgeting for taxes, consultation, and so on. These necessities are required, or would be extremely helpful. Foundations like the Linux Foundation are good at this side of things since they directly manage this work.

The discussion ended with Project Managers: in Node.js, this has historically been something that the project has wanted to see collaborators help with. The group discussed open source and project management, specifically focusing on hiring one with funding. Nobody knew of a successful case of OSS hiring a project manager since project managers have typically been hired from outside of the community, rather than from within.

## Takeaways

- Public and transparent funding management was generally agreed on as being the best path, with the caveat of privacy when required/ in the case of safety.
- It is vital to have clear, concise, and enforceable rules around how funding can and should be used.
- Taxes and other fees being managed is important. If they are not well-managed, this adds to the burden and risk of maintenance.
- Having the ability to hire a project manager is something people would like, but have not yet seen a well-defined path to success for.

# *Velocity vs. usability in OSS*

Participants shared ways to preserve usability, but still ship fast enough. The discussion started slowly as many attendees did not arrive until a few minutes into the conversation. As a result, the topic was explained multiple times. There were no conflicting views in the conversation and it turned out to be mostly a "share and tell" about different tools, plus a substantial conversation about communication.

The topic focus was eventually summed up as relating to how the "lack of dogfooding" shifts the focus toward velocity shipping, which hurts usability. Without dogfooding, it can be hard to see design flaws. There is a point where the developers have a limited understanding of usability because they are not using products in a way that their users are. One participant commented:

> "The moment I get lazy, there is a usability problem."

A participant shared that if they notice that they are typing up a lot of help functions, then they will file an issue to improve the developer experience. Unconscious bias helps maintainers to work around the limitations, but this creates problems: maintainers know where to not step and so avoid the hard problems. After awhile you stop finding bugs and you learn how to avoid them. You become immune to new bugs and seeing them. Maintainers might not realize that an issue comes up more and more:

> "What is the process for empowering users to have the courage to say that the docs are wrong?"

The participants gave examples of how they communicate to contributors, i.e. mailing lists, email, blog posts, etc. Most attendees gave examples for ways they communicate.

Some people find emails to be very formal, and if their email box fills up it causes anxiety. Also, for some people GitHub issues feel similar to emails. Most participants mentioned they prefer not to use emails. They do not mind receiving one-way communication with lots of information, like a mailing list. Email causes anxiety as maintainers have an overwhelming amount of email and do not find this form of communication manageable. On the other hand, emails can be a great way to discuss delicate matters.

Mailing lists and Slack can create additional burdens. Kubernetes had the clearest structure and seemed to impress the rest. IRC can be scary for some people too.

There was discussion about how to manage communications, and what is the ultimate source of truth if participants are communicating across multiple channels. A participant commented:

> "GitHub is the source of truth because everyone has GitHub."

In practice what this means is ensuring that the important messages communicated in different channels are amalgamated in GitHub.

## Takeaways

- Watch people use your project: anywhere they get stuck, there is an obvious design problem.
- Using students to test your project is also helpful to discover design problems.
- Synchronous communication is helpful to help find out the weird things about your project.
- Be aware that different people have different communication preferences.

# *Distributing donations*

This space had an alternative title, “We have money in our Open Collective, how do we use it?” There were four attendees, including a participant who has a funded open collective (they ended up becoming the group leader), while the rest hope to be funded one day or support projects to raise funding. The conversation turned more into a solution and information gathering opportunity for the group leader.

One thing that came across loud and clear from every maintainer is that it is important to recognize that maintaining is real work and contributions from collaborators should be rewarded in some kind of fashion, so that is very important to the community as well.

The proposed suggestions to use can be described as being past-oriented and future-oriented. TPast-oriented activities include rewarding the developers in the form of reimbursement or payment for the work they have done. There were lots of questions about reimbursements, about how to even begin to decide what fair pay is for some of the work, and hesitations around dealing with money itself in an open source project (this can become really complicated). These include questions about the project itself and logistics of the maintaining of the project, e.g. how many members are in your core team, outsourcing of any work, did you ever pay anyone for anything, etc.

In the future-oriented activities discussion, one maintainer’s suggestion was especially well received by the others: to focus on the future rather than contributions of the past and rewarding the maintainers who have done work previously. This is because when they did the past work the expectation was their love of the OS project, not monetary motivation. So perhaps it would be a bit unfair to go back.

## Takeaways

- Establish some kind of bounty program or a feature program where if there is a certain feature that you want implemented on your project there could be a reward for the developer who carries it through.
- Funding meetups or lightning talks.

- Spending for swag, flag labels, t-shirts.
- Maintainer work is real work, therefore rewards are essential to the community.
- The use of funds can be past-oriented or future-oriented.
- Past-oriented means to reimburse contributions, but requires an audit of the project such as membership, outsourcing of labour, and any expenses or payments made. The definition of fair work must be resolved.
- Future-oriented examples include establishing a bounty program, funding program features to be built, funding meet-ups, lightning talks, swag.

# *Schooling maintainers & contributors*

This discussion was about how to help to build up new maintainers and contributors.

New people need guidance and a vision they can follow. There isn't one way that fits all; different projects have different requirements and needs. Sometimes projects are very complex and it's difficult to find a good way in. Also the constant jumping around between communication and writing code can be stressful.

There is agreement that working on open source can improve communication and collaboration skills. And there seems to be a lot of enthusiasm and people wanting to contribute, but it's not always clear where and how to start.

One good way could be to add "how to contribute to open source" to coding schools curricula. This would also involve teaching communication and leadership skills. Open source projects could work directly with schools based on their contribution needs.

Another way could be mentorship so that future leaders can learn from existing ones. When the previous leader departs, the trainees can take over and carry the baton forward. In such a constellation, it's important to have a solid foundation on which to base the leadership. With a good foundation everyone can develop their own style, but stay in sync with past and future leaders. This also ensures that people can look up to role models to help them be good leaders.

Participants discussed creating a "School of Maintainers for Maintainers". This could be a masterclass as part of a Maintainerati event. The ideal would be that leaders talk about their daily experiences and problems they encounter and let other people learn from it first hand.

The group defined the following qualities for a good leader:

- Be willing to take on responsibility.
- Be willing to learn.

- Have empathy.
- Have charisma.

Also, case studies might be a useful and easy accessible tool to document and learn from previous experiences.

## Takeaways

- Have a documented, solid foundation for a project (vision, Code of Conduct, etc.).
- Reach out to coding schools and see whether they want to collaborate and add your project to their curricula.
- Work hard to replace yourself and mentor new maintainers.
- If you are a maintainer, be willing and open to share your positive and negative experiences so that others can learn from them (in a workshop, as a case study, etc.).

# *Maintainability: Getting people doing it*

This session focused on maintainability and technical debt. The problem discussed was that adding new features is more exciting than working on technical debt. There were three themes in this session. The first one, which was the dominant conversation, was the question of maintaining versus working on creating new features. The next was how to create maintenance cultures and platforms, and finally, the third was issues relating to membership and participation within these communities.

## Creating versus maintaining: A moot argument

The first part looked at the issue as two separate activities. Therefore, the main problem was seeing creating and maintaining as two opposing activities. The common question was:

> "How do we get people motivated to work on things like refactoring and bug fixing when there is so much enticement to create new features?"

One participant disagreed. With their project it was really hard to get people to build new features or observe deprecation policies or issue them, because in their project it takes a lot of careful planning to create a new feature that breaks things or that requires documentation because they are very policy-heavy. Instead, people would much rather work on maintaining what's already there and improving it, because that's the easy, low-hanging fruit.

In contrast, the participant who posed the earlier question had difficulty convincing people to work on technical debt. They were asked, "Are the people who are building new features motivated to maintain them over time?" The collective response was "not really". Here, their answers showed the strong relationship between creating new features and the difficulty of maintaining them.

One answer identified was that the lack of interest in maintenance is due to the fact that:

> "Their tests are a mess. It's a cross-platform effort, lots of tests are disabled for certain corner cases because you can't build a feature on a certain platform and now there's a culture of disabling tests and messing with tests, which creates more instability."

One participant identified the root cause as being that:

> "Disabling tests creates debt and pulling things out of limbo almost never happens. You need to have a zero tolerance policy for this kind of behaviour, or at least something close to that."

Another root cause for instability is frequently that:

> "The project is broken into a set of discrete modules. Module API changes happen without updating the modules that depend on it, just the docs. This is a real problem that is created by the fact that the project is so big that it cannot be compiled at once. You cannot create integration tests over the whole thing."

Another participant gave a third reason for an unstable system:

> "Finding module dependencies manually is an extremely time-consuming and error-prone behaviour, so people don't do it. The result is that people are afraid to make new features because they don't want to go through these processes and they don't want to face the consequences if they fail to go through these processes."

However, another participant pointed out that"CI tests are not actually an appropriate place for an API contract, and that heavy process exacerbates laziness." Instead, the participant questions how to improve this system. "There has to be some other way to ensure that API changes are not breaking changes, or when they are breaking changes things get updated."

## Some solutions

There were discussions about potential workarounds, particularly how to avoid, "unintended uses." Do you build policies around accepting unintended uses? One commented that this creates an unwieldy process and formalises bad hacks.

On the other hand, do you just tell people "tough luck for you", when they use your API in a way that was not intended, and breaks when you update it? This answer:

> "Does not generally generate a lot of good will inside your organization. Moreover, deprecations are never trivial, and you need to take time to ensure that users are well supported, so it's a more heavy process."

One participant mentioned that, internally, if you make a breaking change in the code base at their company, you are responsible for fixing code that depends upon the old behaviour.

All participants agreed that this was an interesting sort of way to handle this:

> "There is no process for deprecating features; there is no process for going out and proactively fixing things, but if somebody comes back and complains then it's on you to fix it."

However, there is cautionary advice from this solution even if the other desired outcome includes a "version of really big features and more careful rollout planning." However, these "can lead to large team failures, when necessary breaking changes can't get buy-in, or break too much. Then the team responsible for these changes ends up breaking up over stress. It's not clear if this is exactly the right thing for open source projects."

So, the key issue is, how do we communicate:

> "When a change breaks things; for example, if you don't have a Windows machine, checking changes against Windows doesn't happen. You may not even be aware of the fact that you are making a breaking change. So, if you can't support a platform, deprecate it. Of course, that's not always possible; you can't willy-nilly deprecate Windows, but there is that thought."

While the solutions were not clear, the bigger questions that interested everyone were:

- How do you create a culture of developers on a project who care about testing and supporting platforms?

- Who cares about developing features that work on all supported platforms?
- Who care about the deprecation process when it's time to do that?

Ultimately, although the group were talking about technical solutions to this problem, they came around to the idea that this is a human and cultural problem.

At that point the conversation took a turn to the question, "When do you stay on a team?" You don't stay on a team that makes you unhappy even if you really like the tech. At that point the participants began to discuss terrible teams that they have been on in the past, and they decided to end the conversation at that point.

There was one last observation at the end that brought everything back together, that only more senior people talk about interpersonal issues because it's much harder to have those conversations, and so perhaps one thing that could be done in the future is have a mentoring program for junior engineers to help them understand how to bring up interpersonal issues in a way that is constructive, rather than have them hide behind the idea of "Oh, there's a technical solution to this problem somewhere, we'll ignore the human element to it."

## Takeaways

- Creating and maintaining are intimately related and creating solutions should take account of two aspects of feature design and who and how maintains that feature.
- A communication protocol must be in place or discussed to communicate when a feature breaks. Otherwise, teams split up.
- Ultimately, how do you create a culture of developers that cares about testing and supporting features? How do you keep them happy working in a team?
- Seniors tend to talk more about interpersonal issues. Create a mentoring system for seniors to juniors to help teach how to bring up constructive discussions on interpersonal issues rather than attribute it to technical causes.

# *OSS project marketing*

In this session, the participants discussed project marketing, both from the sides of smaller projects looking to grow and bigger projects that are already well-known and used.

One of the first and most important things discussed was being genuine. We, as developers, and those developers using our projects, generally push back on anything that doesn't smell right.

One example of being genuine that was brought up is Kelsey Hightower: he is in the ecosystem, telling people not to use Kubernetes if it's not a first for their problem… because it's objectively not a fit for every problem.

Similarly, React solves Facebook's problems but is not necessarily the answer to every problem. Moreover, there are React developers who will go beyond the hype and admit that, and tell you that you probably don't need React for every project.

Those who work on/maintain well known or popular projects had some insights that they shared with those who were gaining traction for their open source projects: there are benefits to being the only user. Having a massively used project that dozens or hundreds of companies rely on can sound impressive, but it can very rapidly become a burden on your work, your career, and your personal life. Why even publish projects?

> "Open by default" is a good approach and doesn't hurt anyone. Additionally, if this gets baked into corporate culture, it makes a longer-term commitment to open-source more natural.

On corporate open source projects compared to individual open source projects, having a corporate marketing team isn't required. What ends up making projects successful is finding people who are also passionate about the project's goals/purpose and enabling them to find success, belonging, and ownership within the project.

Beyond collaborators, finding a first user is vital to continued success. Having a first user helps answer questions like: Why should this project exist? Is it effectively solving the problems it addresses? Are there better approaches?

Finally, we discussed roadmaps and their utility in encouraging community. The group generally agreed that roadmaps are an excellent tool for well-defined projects that have non-trivial usage. However, building out a roadmap on new/incomplete projects is often over-optimizing for a future that is likely more fluid than a roadmap would indicate. Additionally, they place even more burden on maintainers from an early stage when those maintainers could benefit from fluidity in project direction.

## Takeaways

- When representing a project in any way, being genuine is one of–if not the–most important things you can do.
- Having projects be well-known or widely used can have drawbacks (both for the project and for maintainers) in addition to benefits.
- Having a corporate marketing team isn't required or necessarily beneficial.
- Roadmaps should be created when they're needed, not necessarily as tools to get more engagement with the project. They can limit project flexibility and build out unnecessary structure.

# *ACKNOWLEDGEMENTS*

**Maintainerati Board 2019–2020:**

*President:* Gawain Lynch
*Secretary:* Erin Taylor
*Treasurer:* Don Goodman-Wilson

**Report production:**

Don Goodman-Wilson | Erin Taylor | Gawain Lynch
Alexia Maddox | Melanie Uy | Nicole Weber

**Note-takers:**

Ben Balter | Tierny Cyren | Katie Delfin | Brian Douglas
Don Goodman-Wilson | Andrea Griffiths | Justin Hutchings
Mike Kavouras | Wilhelm Klopp | Mace Ojala | Paul Oliver
Lee Reilly | Erin Taylor | Nicole Weber | Henry Zhu

# *ABOUT THE MAINTAINERATI FOUNDATION*

**The Maintainerati Foundation** supports the maintainers of the global digital infrastructure to build healthy, productive, inclusive, and sustainable communities. We act in collaboration with maintainers and other relevant stakeholders to:

- Collate knowledge about running healthy, productive, inclusive and sustainable communities that produce digital infrastructure.
- Package this knowledge in ways that maintainers can apply it to build healthy, productive, inclusive, and sustainable communities.
- Distribute this knowledge broadly among the community.
- Evaluate our progress in consultation with the community and re-frame our strategy and practices accordingly.

Stichting Maintainerati Foundation is a social benefit foundation incorporated in The Netherlands, KvK number 76011690.

## Contact Us

*Website:* https://maintainerati.org

*Email:* info@maintainerati.org

*Twitter:* @maintainerati

# *EVENT SPONSORS*

Thanks to our event sponsors for making Berlin 2019 possible!

GitHub

open collective