2021

Sustain

Sustain in 2021

Acknowledgements

The authors and organisers would like to thank all of the attendees of the Sustain 2020 event, and all the working group members and contributors to the Sustain community, whose knowledge and experience has once again formed the basis of this report.

No one, single voice has been referenced, quoted or attributed in this document. In keeping with the respectful tradition of creating safe spaces for members of our community we instead present an interpretation and overview in keeping with the spirit of what was said.

With that said we would specifically like to thank our organisers, our sponsors and the authors who contributed to the production of this report and continue to keep Sustain on the road.

Sponsors & Organisers

Sustain 2020 would not be been possible without Support from:

Cloud Native Computing Foundation

Facebook Open Source

Salesforce

Github

Mozilla Foundation

Indeed

Cloudflare

Open Collective

Open Source Collective

And our organisers:

Pia Mancini, Allen ‘Gunner’ Gunn, Alyssa Wright, Justin Dorfman, Richard Littauer, Eric Berry, Cat Allman, Megan Byrd-Sanicki, Danese Cooper.

Authors

This report compiled, edited and designed by: Richard Littauer, Benjamin Nickolls, John Hill, and Martin Wright.

We would like to thank everyone who contributed to this report including: Georgia Bullen, Eriol Fox, Samson Goddy, Bolaji Ayodeji, Peace Ojemeh, Rachel Lawson, Kevin Owocki, Michael Brennan, Dan Blah, Erin McKean, Tony Urso, Stephanie Taylor, Titus Wormer, Greg Bloom, Bobby Norberg, Javier Luis Cálavas Izquierdo, Mala Kumar, Josh Simmons, Henry Zhu, Georg Link, Duane O’Brian, Xavier Damman, Sophia Vargas, and Luis Villa.

To anyone we may have missed and to anyone involved in the Sustain community: we thank you too.

Introduction

Sustain is a community exploring, documenting and *working* to answer a single question:

How do we maintain open source software?

Open source software is at the heart of many vital platforms and technologies. Therefore it's crucial that this software is resilient and effective. That means that the people involved in creating it need to have access to the latest thinking on how to build, maintain, troubleshoot and develop their work.

But the open source community isn't just about helping yourself, and building yourself up. It's also about asking the question: "What help do others need?".

Scratch the surface of an open source project - even a thriving one - and it's not uncommon to encounter a person who's weighed down with requests, or is facing a lack of funding or support. Instead of asking what that person can do to resolve that situation, we choose to ask what *we* can collectively do instead.

Sustain provides shared spaces, online and physical, to those who are concerned with the fragile state and future of highly-used, impactful open source projects.

We recognise that ensuring a sustainable future for these projects is as much about community as it is technology. Our aim is to amplify as many voices as we can, so that we can all work together to make open source better.

So what is this report for?

The last two years have been eventful, chaotic, and often isolating. But many of the issues that we came together to address remain the same, as does our commitment to resolve them.

Our community has always been global, and distributed. We are used to the opportunities - and challenges - of solving problems with people in different countries and time-zones.

But being distributed means it's not always easy to discover what issues are out there, who's tackling them, and how others can either help out, or start something meaningful for others to join.

This report is an attempt to address that. It's a combination of the insights and discussions we've encountered at previous Sustain events (primarily our 2020 Sustain event in Brussels), and what we've learned through our own work and interactions in our shared, digital space.

We've arranged this document so that you can jump in and explore problems at whichever stage most inspires you - whether that's the early stages, building and scaling, or the work involved in passing the project on.

This is not a definitive "state of the nation", or a "how to" guide. It exists to give you a primer of the work that's currently ongoing, to introduce you to people doing interesting work, and to invite you to consider fresh solutions to ongoing problems.

Sometimes we'll offer our thoughts, and sometimes we'll simply set the scene, so you can come up with your own.

Whether this inspires you to collaborate or innovate, we hope that this report provides useful information about where we are, and vital inspiration about where we could go next.

Using open source software

The success of open source software as a movement is an amazing achievement that should be celebrated.

But, at the same time, this success has created a challenging environment for a sub-category of highly-used components, with a small group of experienced maintainers toiling at their centre.

In *Working In Public*, Nadia Eghbal provided us with a metaphor and a vocabulary for characterising these projects: Auditoriums.

Characterised by a stark asymmetry of knowledge and participation, auditoriums are at the centre of conversations about sustainability.

While the remainder of this document is focussed on what maintainers of these auditorium-like projects can do to become more sustainable, this section is focused on what we as consumers - as audience members - can do to lighten the load for the few in the spotlight.

Requesting support

Considering that open source software is provided 'as is', it is surprising how far many maintainers go to support their users.

Maybe it's a mix of professional pride and the philosophy of peer-support and collaboration that leads many to work in open source. But you'll find maintainers spending a lot of time triaging bugs, addressing calls for help and otherwise providing support to those who ask for it.

So what can we do to ensure that - when we do require some assistance - we lower the time commitment and mental load required of those who *choose* to provide it?

Update your software

Let's get the simple stuff out of the way: reporting problems or asking for support in outdated versions of software is likely to be a drain on maintainer time. So before proceeding any further, update your software.

This means:

- Ensuring that your environment is up to date
- Ensuring that your project's dependencies are up to date
- Ensuring that you are using the most up to date version of the package or project causing your issue.

Yes, sometimes there will be constraints. There will be company policies, access rights and other issues. But it is on you and your organisation to solve those issues before placing pressure on the maintainer(s) by making a request.

And yes, sometimes mature projects will offer long term support (LTS) versions of their projects. In those cases, ensure that you are using the most up-to-date minor or patch version (semantic versioning is an accepting standard for indicating compatibility) before filing an issue.

Support yourself and those around you

The rise of open source means there are now more tools out there for people to provide peer support.

Whether it's a direct question and answer format via StackOverflow or a more conversational platform like Gitter, many projects provide avenues for users to support one another.

Participating in these communities can reap benefits beyond getting past an issue. Often ideas, experiences and common pitfalls are shared as part of the discussion. The opportunity to find and build relationships within the community - whether you're doing it as an organisation or as an individual - can also reflect well.

Active and universal participation in peer-support groups can considerably lower pressure on maintainers to repeatedly solve similar issues in varying contexts. They also often provide an environment for learning, and becoming a more active member of the community.

Finally, ensure that the answer you find is adequately documented. Whether that's using the tools preferred by the community, or for your own benefit inside your organisation or team, ensuring that answers to common questions and solutions to common problems are

documented and discoverable is one of the lowest-cost and highest-impact ways of contributing back to open source projects.

Filing good bug reports and support requests

Many mature projects have considered deeply what they are looking for in a bug report or support request.
Often they are looking to you to do some work to enable the maintainer to quickly triage and assign issues as necessary. Most projects like these use issue templates which contain guidance for those who are requesting support. If that is the case we implore you to *follow these templates closely.*

Occasionally you will find a platform or a project that does not yet have a process or guidance for filing good requests. At which point, consider the following points when filing an issue or support request:

- What environment are you operating in? (operating system, version, dependencies etc)
- What steps did you take that resulted in this problem?
- What did you expect to happen?
- What actually happened?
- What steps can someone else take to reproduce this?

If you can, include a stack-trace: the final output of a program before it exits often includes the call-stack at the time of producing the error. That's incredibly valuable for developers, especially if the steps to reproduce the error do not always result in another.

Lower your Expectations

Lastly, when filing a bug or support request, lower any expectations that you might have on the maintainers that you are asking to give it their time and attention.

Maintainers who *choose* to give their time to support their users often have to do so on an impact-per-issue basis.

If your problem is unique, esoteric to your environment or your use case, or otherwise uncommon, do not expect a response. If your issue is common or highly impactful to you and/or your organisation, consider what you can do to support the project and community.

Building open source software

Building a community

When you're taking your first steps with a project, there's a chance that you'll be propelled by something, whether that's a desire to fill a gap or address a specific user need. That means it'll be tempting just to leap into delivering the solution, and leaving the admin for later.

But there are a few things you might want to consider that might help you start your project with a stronger foundation, and provide a better service for your community.

Trust us: taking a bit of time to set all this at the start is going to save you a lot of time and stress down the line.

Born sustainable

The Challenge:

Sustainability is often an afterthought in project development. But considering it from the start can save you problems down the line.

For example, how do you plan for governance changes? How do you ensure that the community stays healthy? How do you plan end-of-life before the first commit?

These aren't just questions you ask and the start and then never talk about again. Even if you're working on a more mature project, they can help you assess whether you're still on track, and how you can improve and refine.

But what questions should you be asking yourself on Day One?

The Born Sustainable Checklist:

It's not always obvious what you need to ask yourself at the start of a project. You'll learn some things from books and research, and others from experience.

At Sustain 2020, Erin McKean facilitated a discussion that aimed to pool everyone's hard-learned lessons, and construct a "checklist" that could help others.

There are two checklists here. The first covers the code and the project governance with a more code-focused bent, while the second focuses on the community as a whole and how the project positions itself for them.

The checklists are designed so that anyone starting out can work through them and rate themselves. But many may be hop-off points for deeper discussion.

Reliability Checklist

- [ ] Is the governance model written out? Is it in an accessible location?
- [ ] Are there regularly scheduled meetings (one a year, or as needed) around updating the governance structure of the project?
- [ ] Is there an appropriate license for the project, given its goals and users?
- [ ] Is the project reliable? How is this shown? (versioning releases, uptime, number of contributors, current users, etc).
- [ ] Is there good test coverage?
- [ ] Is there a good security strategy?
- [ ] Is continuous integration continuous deployment (CICD) set up from the start?
- [ ] Are there docs? Are they comprehensive?
- [ ] Are the docs well-structured?
- [ ] Is there a public roadmap?
- [ ] Is funding addressed in the roadmap?
- [ ] Are there scaling processes delineated?
- [ ] Are there regular check-ins?
- [ ] What is the bus factor of the organization?
  (On a side note, one of the participants suggested naming this the raptor problem, alluding to the odds that a maintainer will be nabbed by a velociraptor. Given the rate of evolution, this is less potentially insensitive than elevator- or bus-factor)
- [ ] Are expectations set clearly regarding how to contribute, and response times and responsibilities of maintainers?

Community Checklist

- [x] Is there a Code of Conduct?
- [x] Are there onboarding docs?
- [x] Is there a delineation of roles, and how new roles are created?
- [x] Is it easy and clear for contributors to know how to support and help the project?
- [x] Are there tutorials or walk-throughs for new users?
- [x] Is there a framework set up to enable and sustain mentorship?
- [x] Is there a designated community manager?
- [x] Are there venues to talk to the community about what they did, and how they did it, so that people can learn from each other?
- [x] Is accessibility considered? Are translations?
- [x] Is there a publicly available outreach and marketing strategy?
- [x] Are there ways for the community to propose meetups or check-ins?
- [x] Are there real-time chat venues for community discussion (Discord, IRC, etc.)?
- [x] Does the project have a social media presence?

How to start well

In addition to the checklist, the group made several recommendations for new projects.

- **Be clear on the goal:** Know your project's uniqueness and its position in the larger ecosystem. Questions projects should ask themselves include:
    - Have you defined your project's definitions and scope?
    - Are there other projects doing what we do, and what is your place in the ecosystem?
    - Do you know who your market is, and how to market the project to others?
    - Do we have an optimal size?
    - What are some KPIs we can use on the way to our goal?
- **Maximize Transparency:** Give rich access so everyone can contribute, make all aspects of project activity and governance observable, and set clear expectations for users.

    Or, to summarize succinctly: as *"Help people know what they don't know they don't know."*

- **Provide clear and detailed contributing instructions:** Enumerating all core processes (contributing, issues, docs, demos, security) in order to surface all the ways people can contribute.

    Suggested best practices included having on-ramps, onboarding documentation, having a pipeline of new contributors and cultivating next-generation contributors through mentorship and guidance.

    It's also important to celebrate contributions and milestones, acknowledge people, and have active maintainers who are accessible, as well as engaged community "evangelists" and a collective ethic of solidarity and taking care of each other.

- **Use CICD (continuous integration continuous deployment):** A similar argument could be made for using TDD (test driven development).

In most cases, it's smart to use tools that help automate any processes that lead to developer burn-out from the beginning. Any amount of time spent on a project doing tasks which could be avoided is almost certainly time wasted, and it is larger, background problems like these which lead to difficulty onboarding new users and retaining maintainers.

Examples of platforms to use for CICD and tests are Travis, CircleCI, or GitHub Actions.

- **Create an inclusive and safe community environment:** A community should always be attracting broader skills and wider geographic representation. It's a vital part of ensuring current members have support, and expanding the universe of possible contributors.

  Best practices include celebrating contributions, minimizing jargon, having a thoughtful and intentional code of conduct and fostering a culture of respect.

  Particular attention should be paid to making space for the "low voices" who are less confident or inclined to participate, encouraging diversity of thought, and reassuring everyone that taking risks is okay and contributors should not be afraid to fail.

- **Focus on accessibility:** Structure and communicate the project so people can see how they can participate, and don't fall into the trap of thinking they are a burden based on their ability.
- **Have financial planning in place from the outset:** Ensure you have the budget to cover meetups, resources and other ongoing costs. Identify and diversify sources of funding, and engage with them to develop mutually beneficial relationships.

What's next, and how can I help?

The checklists and advice are a good distillation of some of the important points you might want to consider. But any list like this needs updating, reviewing and refining.

Is there anything you would add, based on your experience? Would you amend any of the points raised?

During the discussion group in Brussels, the group suggested these future avenues for anyone considering working on this topic:

- Making a definition for a SUSTAIN.md file for projects just getting started, which lists the information that would answer the checklist. This would also be related to the nascent FUNDING.md file.
- Drafting a self-serve online questionnaire for you to fill out for your project, similar to the checklist above.
- Potentially writing an open source handbook, including examples, interviews, links, and a bibliography. Such a resource could be posted on GitHub along with a list of questions to ask yourself with supporting materials.

While there is not an active Working Group in this area, there may be community members who are interested in building on this work. If you are interested in working or collaborating in this field, check in with the community on Discourse.

Good docs

The Challenge:

Every project needs good docs! Documentation - viewed from 10,000 feet - is the process of explaining to outsiders what is known to insiders. Good documentation is a tool for onboard new users, but also a tool for bringing users who are already in the community further along in their understanding of how the community works, and how they can get involved.

If the docs are going to be relevant, they have to be written with the reader in mind. But what do we mean by "docs"? Who should they be *for*? And what is the best practice for writing docs for different audiences, whether they're beginners, contributors, documentarians, and sponsors?

Audience Personas:

There are several ways to get an idea of who is reading documentation. Either poll or survey readers directly, or write out a list of basic personas.

In a "Good Docs" discussion at Sustain 2020, a working group suggested these groups as the main doc users:

- Beginners to a project or to the context of the project
- Employees working with open source as part of their job
- Developers
- Contributors
- Users of the software
- Sponsors
- Designers
- Other documentarians
- Search engines

Some of these groups overlap; for instance, with open source, it's almost a given that most contributors will be developers. However, each group has different needs.

The goal of good documentation is to increase engagement and to help sustain the project. For designers, simply writing a list of APIs for a module is not useful documentation. They need information about how design can interface with the project.

Likewise, search engines require their own set of documentation - like the metadata implemented in the Open Graph protocol, or keywords, or a robots.txt file.

Even at this most basic level, you're starting to think about the broad groups that are reading your docs, and what they need. And the more you're willing to consider your users, the better and more useful your docs will be.

Identify the key types of users of your project. For each, build out a representative persona, which may look something like this:

> *"This is Sally Takemoto. She works on open source during her day job at a bank in Atlanta. She's interested in becoming better at Golang, particularly around using different threading libraries. She likes dogs and not cats."*

Are this person's needs being met by your docs? Is there any information missing, or is the information being presented in a way they can easily understand and act on?

Is this person your ideal user? If not, why not? And - if they aren't - who is?

Common challenges when creating good docs

Richard Feynman once quipped of a hard problem that "I couldn't reduce it to the freshman level. That means we really don't understand it."

In the same way, much of what is written about code depends upon the users having a high amount of context that they may not have, and breaking this down for newcomers is increasingly difficult as any project grows in size.

Part of the problem of simplicity is that ordering and organizing information is a difficult task. For one, the documentation must be easily navigable as well as easily understood, and those needs are not always complementary. Further, sign-posting is difficult. It often feels counterintuitive, especially as redundancy and duplicative docs are not easily maintained and may be confusing to your audience.

As soon as code is written, it starts to atrophy. Projects change perspective, maintainers move on, dependencies break. Documentation is no different, and arguably worse; it's never a 1-to-1 mapping to the code functionality, and as such it's easy for details to be

lost as soon as the implementation changes. The only known solution to this, unfortunately, is “constant vigilance”.

But you can make your docs less stressful for your audience. Sit down and think about how a new user might approach your code. Explain how they might do what they need to do, in the simplest possible terms.

Do you think your user now understands everything they need to? What might still be unclear? How else could you phrase your words to make them clearer? Is there anything you could add that might help, such as examples, case studies, visual info, or other explainers?

Like code, the best doc PRs can often remove something - a broken structure, and unhelpful paragraph - instead of adding lines.

If you’re struggling to get into the heads of your users, why not ask for help? Plan documentation sessions, and invite others who may have a fresh perspective. This could also be a great way to attract new maintainers, and keep your documents fresh.

Writing beginner-friendly docs

So, what can one do to make docs easier for beginners?

- **Explain where to find information:** A newcomer often lands on a README, and may see more docs in the file, but may not know where to go. This is an opportunity for tutorials, dictionaries, glossaries, and wikis to really shine.

  These tools can help guide a user along set pathways for digesting information, and can help them out when they need to look something up. Which tool will work depends upon the project. But starting with "here are the things you should know about our docs" is always a good place to begin. (If a glossary is needed for the README, you're probably using unnecessary jargon).

- **Include all of the usual doc suspects:** What's beyond the README and the docs/ folder? The rest of the metadata. Include files like Contributing.md, Security.md, and Sponsors.md (or, if you're an .rst person, that, too. Choose what's best for your project). If you can, include a website that lists more information. Write a

description in all of the manifests: the GitHub online description and keywords, or in the `package.json`, or elsewhere.

The Contributing Guide is particularly important, because it helps a user figure out how welcoming you are. List all of your communication channels, and what the best way to open an issue or submit a PR is. If possible, tailor these to the project itself by giving examples.

- **Show how the code works:** One of the best things you can do in documentation to make your project more approachable, and ultimately more sustainable, is meet the user where they are. Don't just tell them what to do, but show them how to do it.

  Usage examples go a long way towards encouraging adoption. If possible, include use cases that a user can try without having to download the project. Add code snippets. Show off examples of your code being used around the web.

- **Test your methods:** Once you've done the above, test your docs. Have naïve users try them out. Try explaining the code to your parents, using only what's written. Implement friction logs, if you can (Google Analytics works well for seeing where people drop off). Try and ensure that your docs are being used.

- **Copy from the best:** This doesn't mean “plagiarize”. It means “learn from others”. Take a look at projects that are similar to yours and look at how they write their docs. Use popular frameworks that have the functionality you need, like ReadTheDocs.

Writing docs for other contributors

Other contributors have different needs:

- **Show how to interface with the maintainers:** Add bug reports and pull request templates. Make it easy for people to log issues. Add your email, if that will help users know who to ask, and if it won't negatively affect your coding ability. Talk about how your labeling system works.
  Mention when you're available for questions, and when you're not, and what a sustainable time expectation looks like. Set expectations around support.

- **Show more complicated examples:** For instance, show how to set up the environment that is needed to run your module, and what contexts it does well in.
- **Explain your decisions:** We don't just mean governance here; talk about where you made decisions that ended up in the land of technical debt. Explain why you've decided to not support A, but you have chosen to support B. Explain why you want your code to be linted the way it is, and what the commit standards are, and why.
- **Share your goals:** Make a public roadmap. Craft a mission statement. Bring others in to help. These will help show contributors that you're aligned with their work, too. This will also help limit scope creep, by setting clear expectations for what you want.
- **Enroll the users:** Set up a CLA to make it clear who owns your code. Acknowledge users who have contributed in any way to your project. Make them feel welcomed.

Contributing to docs for other documentarians

If you're lucky enough to have people in your project who enjoy writing docs, get them involved, too. Some of your users and contributors might be curious newcomers, who keep running into documentation issues.

So, describe where the docs exist, first; and then describe somewhere how to contribute. Are PRs for spelling errors welcome? Say so! A good spot for this is the Contributing doc, or the Contributing section of a README.

For more advanced users, build a style guide for how you want your documentation to look. Explain your link system, if you have one. Explain the system that builds out the docs if it involves building a website. Explain how users can run these environments locally, and how to bug-fix their docs as they go. Link to the markdown spec. Set up committees, if your project is large enough, to discuss all of the issues with a18y and i18n. These things matter for some users.

Writing for sponsors

Sometimes, the users are actually people we want to help pay us. These users have their own needs. Documentation for them may not be public, but rather privately shareable. Whether this counts as documentation is debatable. In any event, these are good things to add to your docs for people interested in seeing the impact of your code (which may include devs deciding whether or not to use your project):

- **Metrics:** How many people download your project? Can you share that

somewhere?

- **Donate button:** This could be a link to Patreon, OpenCollective, or GitHub Sponsors, too.
- **Expense policies:** How should people expense work for your project? What's that pathway look like?
- **Expectation setting:** What are you doing to shore up CLA issue?
- **Success stories:** Write a blog! Include what you have done with the money, if you've already been granted some.
- **Find a way to tell people that you're looking for sponsors:** Sometimes a big sign saying "Looking for sponsors", flanked by two unicorns, works. Other times, you may want to go with something more subtle.

What's next, and how can I help?

A Working Group on creating good docs has remained active since the Sustain 2020 Summit, with meetings occurring roughly once a month since March 2020.

At the time of publication, the Good Docs Working Group is currently exploring ways to support other Sustain OSS Working Groups by bringing a documentation perspective to active conversations already happening in the wider community.

Governance Readiness

The Challenge:

Governance is one of the hardest problems for open source projects to tackle. For projects which are single-person repositories on GitHub, it's easy to figure out the governance model: the person writing the code controls the code.

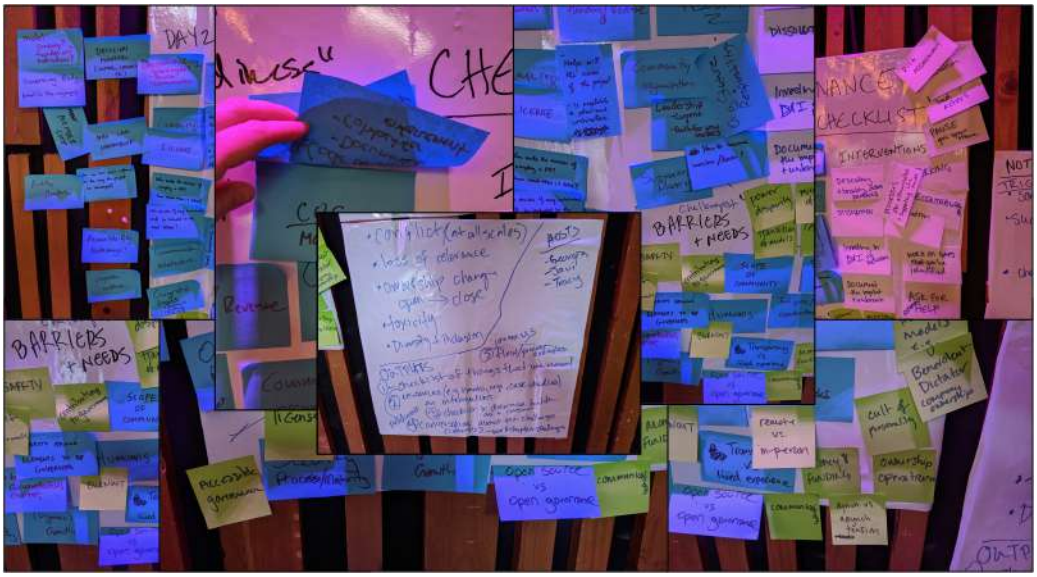
However, as soon as another person commits to the code, governance can become a thorny topic. Who is in charge of the direction of the project? Who mediates conflicts? How are expectations set and met?

Governance is the art of enshrining these social contracts in a clear way, so that everyone can agree on a way of working and get on with writing the code itself. As each project is different, there isn't a single solution to how governance should be handled.

But, as with so many things, it’s definitely worth thinking of these questions in advance, to avoid confusion later.

The Governance Readiness Checklist:

At Sustain 2020, a working group met to discuss what governance readiness might look like. The discussion was focused on idea generation at first. This resulted in a lot of sticky notes, far too many to fit into any cohesive narrative.



However, this work continued after the event. The Governance Readiness working group aimed to define a methodology for defining healthy governance.

As part of this work, they created the Governance Readiness Checklist. The checklist focuses on asking questions that can be used by readers to illuminate gaps in their current governance models.

Below, we reproduce the top five points for each of the three dimensions, to give you a flavour of what you might want to address in your own projects. These questions don't provide answers. Instead, they highlight where the gaps may be in your own documentation.

If you're looking for a more comprehensive list, check out the checklist online.

Day-to-Day

This dimension explores questions regarding governance that you may have to face when developing OSS projects.

- **Which decision-making model should be deployed?** Do you plan to make decisions on your own? Or maybe counting on the support of some kind of committee? This question is one of the most important day-to-day issues you have to face regarding governance.
- **How did you set the governing "body"? And what is it?** The term "governing body" may be scary, but the point is to think about where decision-making power is concentrated, and the support needed. Do you visualize it as a light-weight, or do you need organizational support?
- **Who should define the decision-making model?** This question makes you think about the actors in charge of creating (or evolving) the governance model. It may also trigger some discussion about how to do it.
- **Who can have some influence on the way the project is developed?** This question specifically refers to those situations where collaborators want to have some impact in the way your project is developed. Think about both project developers and external contributors - can anyone have a word on the way the project is governed?
- **Who makes the decision of accepting a pull request? How much time does it take?** Think about the situation: your project receives a pull request from an

external collaborator. Who will review it, and accept it? Do you have some time framing policy for treating pull requests?

Barriers and Needs

We know that things never go as we initially planned. This dimension includes the main challenges that you may have to face when dealing with governance definition and management.

- **Is your governance model easy to access?** Think how difficult it is for newcomers to understand how your project is governed. For instance, how hard is it for newcomers to discover who is behind the project leadership?
- **Have you considered the organic growth of your project?** As any other project under development, your project will evolve over time. Have you planned how to deal with that in terms of decision-making mechanisms?
- **Has anyone in your project experienced burnout?** Take care of your contributors, they may feel they are working too much in your project. Paying attention to the health of your community is crucial.
- **Have you identified the scope of your community?** The community of any Open Source project can have different faces: are your community members developers with a strong technical background? or are they end-users? It is important to know your community and their limits, as it frames how you interact and plan for their needs.
- **Have you studied whether there is a single point of failure in your project?** This is an important task to do: try to discover if your project can fail due to a single point-of-failure (ie. component, developer, etc).

Interventions

Sometimes we need a last resort solution. This dimension explores the possible actions to apply changes related to governance issues in OSS projects. In general, this question assumes you've either had an incident, or you are planning for one.

- **How can you remove bad actors?** Sometimes the best solution is to isolate or remove the contributors that are causing problems. How does your organization do this? Have you removed any you know shouldn't be there?

- **How can you apply pauses or time-constraints in your project?** Sometimes time is the best solution, think whether some time (or space) could make the problem disappear. How would you institute this in your project? Have you, in times of need?
- **Have you tried deescalating, or breaking down the problem?** Being proactive is often the best way to fix the problem. Analyze and face the situation to really understand how to solve it!
- **Sometimes the problem is in the implicit unknown - could you document and make it explicit?** Sometimes we oversee the actual source of the problem, which may be related with implicit information in your project.
- **Another possible resort: Ask for (external) help!** If you and your community cannot solve the problem, search for help. Sometimes it is the best way to find a different view of the problem.

What's next, and how can I help?

This work above was largely led by Javier Canovas, who would almost certainly be interested in taking it further. Get in touch with him if you have any ideas for additions, amendments, and a fuller discussion of how governance should work. For more questions, consult the Governance Readiness website.

If you've already gone through these questions, and want more advice on how to implement governance solutions, the work of Nathan Schneider and the Media Enterprise Design Lab at CU Boulder is easy to use and informative. Check out Community Rule, their website on getting started with understanding different types of governance.

Transitioning from new to mature

The Challenge:

Heraclitus quipped that you can never step in the same river twice, which is to say that change is inevitable.

But sustainability isn't just about maintaining the status quo. It's about understanding what should evolve, and what needs to remain.

By looking at what defines nascent and mature projects, we can determine how to let projects grow and change without losing sight of what they are. Project maintainers, managers, and creators can then use these insights to influence their own project direction.

Governance and Accountability

Small projects are fluid, and that's both their strength and their weakness.

It's relatively easy to tear down code and to refactor when there are only a hundred lines. This mutability extends beyond the codebase. Generally, getting something done on a small project doesn't require bureaucracy. Instead, processes arise organically which allow repeated tasks or interactions to be abstracted and ritualized.

But every successful project eventually switches from organic governance to established governance patterns. This can simply be the process of outlining how consensus is reached. Almost every new project passes through the tyranny of structurelessness, where a small group elite control how decisions are made. Mature organizations have done the work to figure this out, by identifying stakeholders, polling their community, and instituting blocks on power and dispersing responsibility. This lends itself, in turn, to more stability in the long run.

An early goal for any project, then, would be to survive its own founders. This involves identifying stakeholders as the project grows.

Small projects answer to customers, users, and contributors. And in the first instance, all these roles are carried out by a single or small group of people, and to some extent donors (even if it's only those that donate time).

Larger projects have a much more diverse set of stakeholders, such as shareholders, boards, and collaborators. Naturally, this involves a larger oversight process.

It's always worth drawing up a clear map of stakeholders, even for nascent projects. This can help identify how a project should grow. Decide who should have responsibilities for the project.

Culture

It is impossible to speak of stakeholders without thinking of the culture of a project and its community.

A small project will focus on early adopters and early users, who are willing to jump through hoops to use a codebase if the utility function is high enough. Larger projects tend to do more user experience testing, to lower the barriers to entry so that it's used by as many people as possible.

In the same way, the team needs to grow, from "genius tinkerers" to more diverse groups that are able to see the problem the code solves from multiple perspectives.

And as the scope of a project expands, so do the needs of the users. This inevitably slows down development, leading to a need for larger development teams and, again, more process.

With larger teams, other needs surface. For example:

- How to gracefully handover reins to new maintainers
- How to maintain projects with users of different levels of experience
- How to onboard new users and new team members
- How to honor contributors' effort... without succumbing to contributors who take more than they give back.

Training and maintaining these teams takes time, and it's difficult to always make the best hiring or onboarding decisions.

Small projects tend to hire "like themselves", and emphasize culture fit (especially at a project that is also tied to a startup). Larger projects tend to enforce diversity, and have processes where many different types of individuals can contribute.

On an even simpler level, many projects start with all of the staff being volunteers, whereas almost no one "volunteers" at FAANG. Planning for growth and planning for maturity are different tasks that require different considerations.

So, at the earliest possible point, sit down and figure out what your culture is, and how much it is being influenced by the core contributors. Listen to new contributors' needs, and decide how you're going to onboard new maintainers to foster a better culture.

Funding

Money, of course, is the difference between a volunteer and an employee.

Small projects often have little money, and few tools for getting some. Some wildly successful projects in the development space, in terms of adoption, only have "sticker money" (to quote Mike McQuaid) or subsist entirely on grants, like cUrl did with Mozilla. The ability to market a project is often what makes or breaks a project in terms of sustainability.

At the same time, large projects also have more costs, higher staff turnover, and the needs of different funders to consider. They also have to maintain brand awareness, satisfy regular sponsors, and consider how to keep developers on the project.

Nascent projects are able to avoid this by depending on donated time or smaller budgets, and have the ability to adroitly change their mission to find an ideal audience or market. It's easier to pivot without arthritis.

Decide early what kind of funding is best for the long-term viability of a new project, and act accordingly. Undertake a frank appraisal of various funding schemes before deciding on a viable future that doesn't involve maintainer burn-out.

Vision

Pivoting involves a change of vision.

For this, small projects are again in an entirely different situation to larger projects. A smaller project can define their vision clearly and succinctly, and often don't need to justify it to stakeholders or to their audience. This makes it easier to change to suit the needs of the project.

A large project, on the other hand, needs to both have a vision that is vague enough to suit all possible uses of the project, and needs to be strong enough to be passed on to the new members of the community easily. Existential crises raised internally by having users such as DARPA or ICE are less likely to occur to smaller projects that have been able to flow beneath the radar.

Vision is more than just a mission statement, however; it can also entail goals for the project. Smaller projects can find it easier to limit scope creep or bikeshedding, in particular by leaning on the judgement of a BDFL or a small cabal of interested users. Larger projects may not have this ability, and are often constrained less by internal vision than by bureaucratic processes, such as strict prioritization of possible features.

Define what your vision is for your project, and how you will bend the roadmap around this vision.

What's next, and how can I help?

Navigating the road from a small project to a mature one is tough, and each project will do it differently. The difference was eloquently summed up at the session by the difference between "How would we like to work" and "We've always done it this way".

Discussions around this topic at Sustain 2020 did not lead to the formation of a working group. However, there is always an opportunity to further the community's knowledge on this subject, leading to more informed and diverse discussions.

This could involve sharing your experiences of growth and change, and highlighting issues and advice that might be valuable. There is also value in chronicling and curating stories of projects that have encountered interesting challenges, and navigated them successfully (or unsuccessfully).

These stories could be shared on the Sustain forum, or by a stand-alone project.

Maintaining a community

Before you know it, you're managing a community. Maybe that's a small group of people, or maybe that idea you had has inspired a whole lot of people.

So how do you ensure you keep the momentum going? In this section, we'll discuss a few helpful themes that have emerged from past projects, such as how you might resolve conflicts, demonstrate good empowering leadership, and minimise the effects of burnout.

Governance principles (ostrom)

The Challenge:

Working within a community with common interests and ideals can be empowering and exhilarating. But how do you ensure you effectively cultivate and protect systems which are shared?

In *Governing the Commons: The Evolution of Institutions for Collective Action* (1990) Elinor Ostrom set out a set of principles by which a commons could be governed, and a set of strategies that could be used to better understand a community of interest.

Ostrom believed that there was no one, single solution to solving commons governance issues, that each community needed the freedom to define, police and enforce restrictions. Her work was initially focussed on tangible shared resources, like fisheries, but she later thought about how these principles might be brought to govern knowledge goods.

Principles for Open Source Governance:

At previous Sustain events, attendees have attempted to translate Ostrom's principles to open source software and the communities that sustain it. In 2020, the goal was to revisit, revise and — if possible — complete this translation. Over the course of two sessions the group formalised the following statements of intent:

Clearly Defined Boundaries

We clearly articulate the purpose of our project (i.e. our vision, mission, and scope) as well as our values and principles (i.e. the good we manifest in the world, and the way we do our work). We are guided by our purpose and principles in all that we do.

Appropriate Rules

We make agreements that establish fair relationships among contributors, users, our products, and the project itself. We assign licenses that align with our purpose and values. We set clear expectations for contributors regarding the nature of their contributions. We clearly document all of the above.

Rule-making processes

We make it clear who can participate in what kind of decision-making, under which circumstances. Our decision-making processes are transparent, and account for the diverse perspectives of our community.

Monitoring

We monitor our adherence to our principles, values, and purpose. We engage in regular testing — of code as well as our assumptions. We share information about the status of our work with the community, and we are committed to improvement when we find it to be needed.

Sanctions

When people act in ways that are out of line with our project's stated principles and purpose, we have a range of effective ways to encourage re-alignment — from gentle admonishment up to removal from the community of those who can't or won't abide by our stated expectations.

Conflict resolution

We are committed to fair resolution of disagreements when they arise. The ability to resolve disagreements is distributed and accessible, so that all participants feel like they can participate in the process of resolving conflicts that they are a part of.

Self-governance

Our projects have autonomy to organize ourselves. We make decisions about how we structure our work and our communities as it is appropriate to our projects' principles, values, and purpose.

Subsidiarity

We embrace the complexity of our projects, which involve multiple parties and various scales, and so we engage in decision-making at various scales with various parties.

The group also identified a set of prompts and questions that stakeholders in open source communities should ask themselves *frequently* when looking to establish and maintain a coherent and sustainable governance framework.

For example:

> *"Have we specified who we will accept code contributions from, and in what form?"*
> *"Who gets to act on behalf of the project?"*

“Who can use the project’s name?”

What’s next, and how can I help?

Following the Sustain conference, the working group continued to refine these principles. They published a first draft in The Principles of Governing Open Source Commons · SustainOSS: exploring sustainability for open source communities later in 2020, and continue to meet once a month.

If you feel you might wish to contribute to this work, you can get involved in this working group through the Sustian website.

Accidental leadership

The Challenge:

Leaders in tech are not hard to find: the term "thought leader" is now about as cliché as wearing a black turtleneck.

It's not hard to find any number of weighty volumes about what leadership means in enterprise, in the military, and in the media. In open source, leadership is more of a vague concept.

There are leaders of longevity - Vint Cerf and Tim Berners-Lee, for instance. There are also leaders of established, large projects with elevated platforms - people like Brendan Eich, Richard Stahlman, and Linus Torvalds. There are also more recent leaders who work in organisations or with projects that have granted them platforms - like Danese Cooper, Sarah Novotny, or Saron Yitbarek.

However, leadership doesn't necessarily mean a huge Twitter following or constant conference appearances. Instead, it can simply involve being the person who merges PRs for a project.

These leaders are not necessarily public figures. For example, how do we differentiate creators of small projects from strategic public advisors of large open source enterprises? And what does leadership mean generally in the context of open source?

What makes an Open Source leader?

At Sustain 2020, Josh Simmons led a group that dived into to tease out meaning from this topic. They asked:

- What lessons can we learn about what makes a leader in open source?
- What makes a *good* leader?
- Are there ways to teach what authentic, powerful, and effective leadership means for an open source project?
- Are there stories we can share on how maintainers ended up ascending to the throne?
- Can we use these stories to help others who may be taking the reins of their suddenly-star-spangled projects?

The group quickly agreed that the biggest issue was to build the awareness and skillset of the individual, as there were other groups to tackle project governance, process and communication specifically.

So what traits, skills and knowledge are needed to become a modern open source leader?

In all, the group listed over forty. And none of these were overt technical skills.

Many were variations around the themes of:

- humility, respect and self-awareness
- being open
- being understanding and an empathetic communicator
- maintaining a clear set of boundaries and expectations to protect the community.

Outside of this general set of good, *human* principles, the group identified some specific traits that were relevant to maintaining a successful open source *product* today:

Direction

Maintainers are typically *the* centre of a project's institutional knowledge. They are most aware of the challenges facing a project, and the opportunities available to it. Curiosity, awareness, consensus building, decision making and the ability to genuinely *listen* when challenged are vital here, alongside clear communication and documentation.

Facilitation

Projects that attract a lot of co-contributors will often find themselves in need of a facilitator: a maintainer that enables others rather than focussing on contributing themselves. Creating and maintaining a good developer/contributor experience through documentation, delegation, arbitration and diplomacy are key skills here.

Outreach and advocacy

While open source projects might lack the multi-million dollar budgets to advertise themselves to users, there are simple marketing strategies that projects can employ. An awareness of branding, design, copywriting, and product management is important for successful open source leaders.

Maintainers may not all be earning their crust on the conference circuit, but the skills we see on stage at events are just as applicable to small group discussions and the 'hallway track' that can help recruit a community of contributors and supporters for a project.

Organisational development

A community's culture will evolve from the leadership, especially when it comes to communication and decision making processes. Creating pathways for existing maintainers to 'exit' a project responsibly is just as important as creating on-ramps for contributors to grow into leadership roles themselves.

What's next, and how can I help?

The group considered a number of resources that could be used to support "accidental leaders" in open source. Many of these still do not exist, and could be extremely useful for members of the community:

- A 'choose your own adventure' leadership guide.
- A gallery of good examples of:
    - Project governance
    - Budget management
    - Fundraising
    - Outreach and advocacy
    - Documentation
- A set of workshops, tutorials, and guides
- A recommended reading, viewing and listening list
- A mentorship program
- A programme for maintainers to announce project deprecation and for others to adopt-a-project

If any of these projects sound intriguing, please consider developing them, or reaching out to members of the community for a possible collaboration. Some members may be able to suggest suitable case studies or other leads.

Managing burnout

The Challenge:

The responsibility of maintaining and sustaining an open source project often falls to a single person or a small group, and the weight of that work can lead to mental and physical strain.

As well as maintaining open communication about the progress of a project, it's also necessary for the people running it to be honest with themselves about how well they're managing these pressures, and what steps they're taking it to maintain their health.

The threat of burnout

Of course, the sustainability of a project is also determined by another factor: Burnout.

Burnout is an old friend to anyone who has been in open source for a considerable period of time. Projects spark with inspiration, and passion fuels the development. But as the project grows, alters and becomes more popular, maintaining it becomes a constant task. In those cases, it's natural for the creator's enthusiasm to wane.

Many projects become "abandonware" due to a lack of incentives for keeping them alive. And many developers, after being involved in open source for years, switch to other hobbies or careers that they find more rewarding in the long term.

People's priorities and ambitions change. And it's not feasible to expect everyone to do the same things forever. But how do we make sure maintainers don't burn out before they're ready to pass on the torch?

Who is a maintainer, anyway?

It's not an easy word to define.

The most obvious definition is "someone who maintains a piece of code" - merging patches, handling issues, and keeping track of the vision for the project. However, there are a lot of different hats that need to be worn, and some maintainers only some of them.

For instance, many maintainers aren't the original project creators; sometimes maintainers only merge relevant PRs and don't bother with issues or keep a handle on the community; some community organizers may not have code domain knowledge, but still work with the

community to help the project grow. All of these are called maintainers, but each persona is slightly different.

Knowing what these personas are is a key to understanding “the work”, and where it might ultimately lead to burnout.

For instance, some code creators may not want to interface with the community at all. Marketing a project or setting up donation funding accounts is often something maintainers don't want to do, as it can be seen as project management.

Someone needs to do this work, however. Abstracting it into different roles can help project maintainers determine what work is unappealing, and from there they can decide how to outsource or minimize it.

Our tools are often very code centric; GitHub, for instance, enshrines the values of code over issues. However, this may be a detriment to the community at large.

For instance, it's hard for designers or PMs to manage with GitHub without knowing some code - but their roles are often just as important for a project's growth. GitHub doesn't show recognition for non-code work (although, arguably, they are starting to implement features for this, like the Community Health tooling).

This can be disincentivizing for people who - if they were engaged - could help lessen the risk of burnout for the coders involved. Trying to work on GitHub as a non-coder can also lead to burnout, due to the constant need to wade upstream against the "coders are kings" mentality.

Another issue with the term "maintainer" is that it puts a lot of focus on the individuals in charge, marking them out as the only people with overall vision. This can mean that community members who don't have clear pathways to become maintainers often don't invest in learning the entire project, but instead work on small things, over and over again.

This sentences the maintainers to a life of constant small interactions that require their (and only their) attention. It’s also a negative for the community members, who aren't incentivized to go beyond how their little bit of a project works. It’s disempowering on both sides, and that can easily lead to burnout.

A culture of Busyness

Maintaining can be packed with “small tasks”.

Dealing with drive-by contributors. Responding to low-context community members. Handling projects which are related to - but not core to - the big project.
Small tasks are often easy to get through, but they don't always lead to new creations or a sense of fulfillment for the developer. Being busy does not always mean being productive.

There are ways to mitigate this. For instance, instituting a “code sabbath”, where one agrees not to code or touch a project on a set day. For collaborators who come from corporate backgrounds, this is often known as the “weekend model”.

However, many open source programmers are hobbyists, and taking a day off won’t always help refill the tank of passion for a project. As one person in the Sustain conversation put it: "There are no weekends in open source."

Notification overload is common, as are pressures such as demanding users asking why a bug hasn't been fixed. Many maintainers feel that "everything is on fire", all the time, and it is rarely fine. During the COVID era, with everyone being remote, the expectation to be online all of the time is even worse than before.

Like the code sabbath, the way to deal with this is by learning when to say “no”. Setting clear expectations around the code can help set boundaries, and limit your exposure to negativity.

Use Processes

Open source is distributed, decentralized, asynchronous, acephalous, and generally a mixed bag of thousands of different types of projects.

As such, the rhythm of a project tends to move to the beat of its own drum.

However, companies generally don’t. They're often colocated, philopatric, synchronous, well-incentivized and orderly. Taking lessons from companies in some areas can be a good move, where possible. And, frequently, the key takeaway is process, or “how a person in a project does what they do”.

In open source, often processes aren't written down or linked. It's rare to see a statement in a repository from a maintainer saying, "I check the issues every alternate Tuesday from

2-4pm Brisbane time." However, there's no reason that this couldn't be the case, and the project may arguably be better for it

As some Silicon Valley hackers are fond of saying: “What doesn't get measured doesn't get improved - and what doesn't get observed doesn't get measured”.

Listing a wet process out in its entirety will make it clear how this process can be used. Of course, it's hard to find people with technical expertise who want to write everything down or take a managerial role. But those actions are often precisely what a project needs.

Once a process is written, it is often easy to share it with someone else or to improve on it. One idea that has come up in group discussions is to have a policy of rotating mundane tasks. A maintainer could be the issue-checker for a couple of months, for instance, instead of constantly only writing better tests. By defining and then assigning and rotating roles, a maintainer has less chance of becoming stale. This would also incentivize the boring tasks, such as triage, labelling, or cleanup.

Another point is that, once a process is written, it's possible to track it to see if a maintainer is actually suffering from burnout.

For instance, time to respond to issues is an oft-mentioned metric for the health of a project. It could also be noted as a metric for the health of a maintainer's level of interest in a project. Noting metrics like this can be useful for spotting burnout before it happens, and then reallocating efforts (with the maintainer’s approval) to make them less likely to get tired of their tasks.

Other ways of fighting burnout

Discussions around burnout have raised a number of other suggestions that might help:

> **Have a community manager**
>
> Hiring, onboarding, cultivating - there are many different ways to get a community manager. In general, finding someone who is interested in interceding in difficult situations (such as Code of Conduct violations), clearly explaining expectations and boundaries, and marketing the project towards other developers and adjacent projects will go a long way.

But community managers are difficult to find, and they are even more difficult to embrace as members of the project without them having the domain knowledge necessary to do their work. Often, the role of a manager may need to be taken by several different maintainers or new contributors, who want to demonstrate their intent to stick with a project. Writing down what those roles are is a first step towards filling them.

Plan for succession.

You won't be doing this forever. So, plan for succession from the get-go. Cultivate other maintainers to take over. Set up committees to do this work for larger projects. For every newcomer who comes back, ask if they are willing to do more for the project.Cultivate relationships, and not just the project. This often goes for life as well as open source.

Cultivate a culture of encouraging others

As well as building good relationships, think about cultivating a culture of encouragement. One project adopted a value set of being "more Canadian", and stressed: "let's be the nicest we can be." While not a perfect simile, it was useful in adjusting moods when responding to issues, answering questions on Gitter, or otherwise interacting with the community.

The fish rots from the head; so, make sure that the maintainers and core community have a culture of sharing and gratefulness, as this will slowly branch out through the community. This isn't just an aphorism - the early days of Rust were almost defined by a sense of welcoming community, which ultimately helped that ecosystem flourish, as many developers sick of issues within JavaScript switched over.

There are ways of encouraging community, too, beyond just being nice. Notice when people post on mailing lists or issues, and when people evangelize the project. Notice people who contribute to GitHub issues. If you can, use bots like all-contributors TK to show contributions for the project. Use badges.

Invest in newcomers cautiously

While every newcomer is potentially a new maintainer, many open source contributions are singletons. Recognize that this is the case, and be mindful of

over-investing in any one contributor. The people who will stick around will stick around, and ultimately not everyone is incentivized to maintain code that isn't theirs.

There are projects that actively state the opposite: Google Summer of Code, First Timers Only, and Hacktoberfest, for instance. While these are useful, they often take more work to maintain than they give back to the project, and some contributors may apply the scattershot method to projects to see what sticks. Be as kind as you can, within limits. And if you find yourself answering the same question over and over, create an FAQ. That's what FAQ stands for, after all.

Pay attention to the details

Burnout stems from the emotional costs of working on a project. Often, these costs are minute, but they build up, and lead to emotional disengagement.

For instance, labels themselves can be triggering. An example is the “chore” label, which is negative use of language. No one wants to do chores (no, really). If we did, we would use the word "really-awesome-fun-stuff" instead to describe vacuuming and dusting. The word "issues" is also often negative. Changing these labels in a project, or recognizing them as small microaggressions, can go a long way towards staving off burnout for a project.

What’s next, and how can I help?

Even though there were enthusiastic discussions around burnout last year at Sustain conference, a working group didn’t emerge. However there are resources out there for developers who are experiencing burnout. If you’re interested in adding them - or even engaging in thoughtful discussion about the issues - we always welcome your thoughts on the Sustain forum.

When it comes to determining metrics for sustainability, The CHAOSS group continue to explore and document metrics across a number of lenses: diversity,

value, risk, evolution and ‘common’ areas of focus. Working groups maintain a contribution process[1] for each area.

The metrics published on CHAOSS’ site at https://chaoss.community/metrics are released on a six month cadance. For more information on getting involved see https://chaoss.community/participate/.

[1] https://chaoss.community/metrics

Sustaining a project financially

In the open source community, you'll find a lot of support and goodwill. But money can be harder to come by.

While many open source developers don't get into this kind of work for the riches, there are several things that you can do to make the project more visible, and therefore potentially improve the chances of it getting the support it needs.

In this section, we'll look into business models and marketing your project. We'll also check in on some recent discussions about how to identify support that might be out there for open source.

Business models

The Challenge:

Making a living as an open source developer is difficult.

Most people who work on open source can be split into two camps:

- Workers who are paid to work on open source software as part of their work as employees
- Those who aren't paid at all, but who do it in their spare time.

Independent developers fit somewhere in between. Segueing from being a developer working on a passion project to becoming someone who is paid for it enough to live sustainably is tough, but possible.

Pathways to Money:

There are several pathways to raising money. Open Collective and other donation-based engines can work well for charismatic projects. But they work best for popular projects, or projects which attract clients who are willing to pay through the Open Collective for development work.

Venture capital works well for projects that have a viable business model. But pitching to investors, figuring out an income stream, and running a business based on open source can be unrealistic. Often, it can be easier to set up a small business around the project, before considering selling or raising money. To do this, analyze the market and understand your project's niche, and work hard with your community to figure out how the work can be funded.

Remember: taking on money fundamentally means scaling the project, taking on responsibilities, and ultimately moving away from the source code. Creating a bigger project and raising money is not always the right call.

However, it can be hard to bootstrap your project to get to the stage where you have enough money to pay developers. Too little money, and you're limited to little more than buying stickers and attending a conference. Too middling means that you can only pay one or two part-time developers.

There are ways to help: plan a conference or meetup and ask for donations; fork the project and develop a freemium model with set features for paying customers; see if dual-licensing will work for users of the product, or prioritized support.

Eventually, all monetization is about value. Figuring out what value you can provide that is adjacent to the project is essential. Once you've done that, find a way for customers to pay easily for that value.

What's next, and how can I help?

The biggest challenge will always be how to promote your project. The act of building a business is not the same as working on the code, which means that this work needs to be both intentional and premeditated. Luckily, as the ecosystem matures, different models are being tested, and more opportunities are emerging to help you capitalize on your open source project.

As ever, the best ways for the community to develop in this area is through support, and guidance. If you have knowledge of organisations that may be able to help, your help is always valuable. And there's always a call for more stories - such as case studies of people that have succeeded in developing sustainable or transformative business models, people who have navigated particular challenges, or even people that have learned important lessons the hard way.

Mapping OS Funding

The Challenge:

What support is out there for Open Source projects?

It’s a huge question, but one that involves a little investigation. There isn’t - at this point - a public overview of funding support from the various entities that might be out there, from government to foundations and private businesses.

While the community often shares details of those that members come across personally, there’s still work to be done to build the data on this kind of support, from monetary grants to companies who might be interested in sponsoring events.

Learning where to look:

The open source ecosystem is largely underfunded. The majority of open source projects on GitHub, for example, are either not in active development or were not released with a funding model.

However, there is money available for some projects. That might be because they’re owned and operated by organizations, or because they're on a donation platform, or because they're supported by grants.

Knowing where this money comes from - and where it goes - is difficult. But that information would be very valuable to many different types of open source participants.

As a maintainer, where do you look for grants? Are grants even viable, or should you look for other market solutions?

There are no centralized resources for showing open source financial contributions, or for comparing the relative worth of these contributions against those from other companies.

As a team at a big company, it is unclear how much the company gives in total, and to which projects. As such, it is hard for developers wishing to donate to pitch up the management stack or to go down the stack to show total metrics for budgeting.

As a sustainer, it is unclear how much and what kind of funds are in the ecosystem, which leads to an unrealistic focus on charismatic, large grants.

What's next, and how can I help?

In a discussion group last year at Sustain, it was decided that the financial information should come from three separate research areas:

- From accessible APIs for donation platforms such as OpenCollective, Patreon, and GitHub Sponsors;
- From crowdsourcing data via maintainers of open source projects and foundations
- From public information given by the entities.

This information could be shared as quarterly reports, to make it more sustainable over time.

While the database would initially only display a small subset of the information, the goal would be that - over time - the community would contribute more, and collaborate on a more robust dataset.

Ultimately, this service would be useful to companies wishing to showcase their contributions to open source communities, for maintainers wishing to know who to approach for funding, and for sustainers to gauge the state of the ecosystem.

This project is currently on hold. But anyone wishing to take on the idea should contact Richard Littauer at richard@burntfen.com, or go to the Sustain Discourse.

Marketing open source projects

The Challenge:

While marketing is often considered to be a four letter word, it's also an essential part of making projects sustainable.

We can disambiguate marketing into two categories:

- Sharing the inherent nature of an open source project
- ...and the act of going out and actively selling it.

In the former case, a project's README or documentation could be considered marketing. Having keywords listed on GitHub renders the project more discoverable.

But building an email list, running a project website, and cold-calling investors is normally considered canonical marketing work.

Both are necessary in open source. Without one, no one will use your project; without the other, scale isn't as remotely accessible, and funding will wither on the vine.

A working group was born at last year's Sustain conference, made up of five sustainers. It discussed possible marketing tools, and how effective they are. Their efforts are still in the early stages, but they did offer some ideas on how to build a marketing presence for your project.

> **Keep it real and don't spam**
>
> Authentic marketing is the only way to build authentic contributions and connections. Bad marketing will make your audiences think of you as shallow, disposable, and crass.
>
> So, think about your users and what their needs are. Don't ping them more than you need to, or more than seems polite. Remember that they are juggling other needs. If your product doesn't solve a need that they have, don't sell it at all - go back and build it over, or build a different project. Spam on the internet is not only pathetic, it's also illegal (at least via email). Do your best to make sure that what you're selling is both what you're selling and what the users actually want.

Establish goals

What do you want to achieve with your marketing efforts? Sprucing up the README and starting a newsletter are fine, unless you're unable to cope with loads of incoming beginners.
Do you want more stars on GitHub, or more watchers? Do you want more contributors, or do you want to turn some contributors into maintainers? Are developers the target market for your code, or should you wrap it up in a service and use the open core model? Do you want large funders, or microdonations? Deciding how you want your project to become sustainable will help you focus your marketing efforts.

Build a brand

Open source projects are products, so identity is important.

There are many different ways of building an identity - logos, editorial style guides, coding styles, commitments to respond to contributors within a set time, referring to cats more than normal, or translating your docs. Each of these will signal different things.

Having a designed logo can indicate the project is more established, and help build a trusted brand. But the important part is to build a brand that's "you", and stick with it.

Monitor social media

Unfortunately, using Twitter and Hacker News (HN) is often a *sine qua non* for open source community management.

Brand monitoring with free services such as F5Bot, TweetDeck, and Google Alerts can help maintainers join discussions quickly to resolve any issues that might arise on Stackoverflow, Reddit, or HN. Depending on the size of the project, the question isn't will this happen, but when.

Being able to respond fluidly, kindly, and quickly is important. Likewise, being responsive on Twitter often leads to collaborations, which engenders more valuable contributions than drive-by contributors. If you can, be proactive by actively posting to Product Hunt, Show HN, Dev.to, Cooper Press Newsletters, or

podcasts that cover similar projects.

Write your surface-level docs

Make sure that your READMEs on GitHub or GitLab have all of the necessary information for new users just learning about the project. Good examples include Forem, Bootstrap, Vue, curl, or all of these.

If you are having difficulties, use a style guide like Standard Readme. Or, ask around for README editing services. Besides this, add in the URL to your website, add keywords, and go the extra mile to make sure that your project looks pretty.

It matters, because often developers will only stay on your page for a couple of seconds before deciding to go with your module. Likewise, fill out your manifests for your package managers, in case the entry point for your repository is on a different platform like NPM.

Honor contributors

One of the best ways to market is to outsource to people who care about your product enough to talk about it themselves.

The best way to get these people in your project is to make them feel like it's their project too. Offer ownership and responsibility, and give verbal high-fives whenever you can. List your core contributors and sponsors on your README (for instance, by using the Open Collective plugin). Share ways that contributors can sponsor your project to help them feel invested (like Patreon, Crypto addresses, Gitcoin Grants, and PayPal), and then thank them for doing so. Mention who committed code in your changelogs. Include members in your project updates. Use tools like All Contributors to list contributions that aren't just code related. Lift others up, first.

Show your social cred

List your contributors and sponsors. And, if you are able to find this information, show which organizations or projects use your project. Ask if you can display their logos on your website or README. Embed unsolicited testimonials on your website. If you can, use those that speak to multiple different use cases for your project.

Above all, ask for permission first when doing any of this, and then thank your users any way you can. Send them stickers, swag, or postcards. As always, be authentic.

Build a presence

Building a static website costs nothing but time today. Spend $10 to $15 on a URL, and hosting is free using GitHub pages, Netlify, Zeit, or any number of providers.

A basic project page can go a long way in making your project more shareable. Likewise, building a newsletter also takes little time. Find some newsletters you admire and copy their structure; or, send letters infrequently when you have something to say, like a major release or a bug or a new collaboration. Emails are the most effective way to reach users; collect them when you can, and use them wisely. Be consistent.

Give back

This is the unwritten rule of open source. Be someone who uses open source as a verb, and not a noun.

Whenever you use a dependency, thank the author. Fix their bugs. Give advice when you can. Host newcomers for Hacktoberfest, or Google Summer of Code. Mark your issues as "First timers only". Don't do this because it's good to have a lot of inexperienced contributors - it isn't. Do it because it shows that you're willing to help others, just as you got to where you are today in coding on the shoulders of others. Do what you can to be an active member of the community. It shows, in the long run.

Have fun

Solve your bugs while coding on Twitch. Give out enamelled logo earrings at your meetup. Make a video of you taking out your compost while also responding to GitHub issues where people make unreasonable demands on your time. Don't be normal. Be different.

Market without marketing

Marketing is tough; it takes guts to ask for other people's time. So, do what you can to make it easier on yourself.

Figure out responsibilities in your project, and who will own your various entry points. Build a basic persona document to help your UX, even if you're just building a simple module. Figure out which segments you want to market too. Then, be authentic, and focus on building the core project itself, even if it means you're not consistent with your newsletters or if you miss some tweets.

Set expectations for what you're able and what you want to do. If you're gung-ho for writing hand-written letters to every contributor, do it. If you only want to tweet about the project once a year on your personal Twitter, that's OK, too. Be honest with yourself, and focus on creating.

What's next, and how can I help?

Marketing is universal. If you've tried to build up a project or business in any walk of life, you've probably had to try your luck at marketing, or worked with someone who has.

That means that we're not short of stories we can tell, or lessons to learn. We're always interested in resources, case studies, advice, or any other way that we can learn from people who have gone before. And any ideas for exciting, new and innovative approaches are welcome too.

Supporting open source software

Understanding open source communities (metrics)

The Challenge:

Companies are becoming more aware of their impact on - and responsibility for - open source software. This is reflected in the growth of Open Source Programme Offices, and associated roles. That's highlighted the need for metrics to help them understand and reflect on their involvement.

Communities like TODO group[2] and projects like CHAOSS[3] are leading the way, exploring what is valuable and what is merely measurement for measurement's sake.

Measuring sustainability:

Contributors will advocate for different metrics, depending on their role and what they want to get out of the project. This was reflected in some of the discussions about measurement at last year's Sustain event:

> **What do different OSS personas care about?**
>
> Open Source Programmes and Community managers often have similar questions that they want to ask. Professional users are often driven by risk, specifically around licence compatibility and commitments. But increasingly they are looking to measure the impact that their organisation is having on the development of a community. There is often a gap in metrics that support decisions for funders. This could be why we frequently see projects attracting even more funding as they begin to see donations rolling in.
>
> **There's conflict to be resolved**
>
> There are power dynamics at play when choosing metrics, and it can result in conflicts between different personas. Commit volumes are often biased toward growth-stage projects while governance patterns — like squash-merging pull requests, a pattern in which many commits are squashed into a single contribution — can have an outsized impact on them. Metrics like these also lead to negative, competitive comparison. The group that met at Sustain agreed that sharing case studies is the way forward here.

[2] https://todogroup.org/
[3] https://chaoss.community/

Pull request to acceptance

There was a definite consensus in the group for collective metrics that measure and incentivise *collective* behaviour. The time between a request for a contribution and the acceptance (or resolution) of that contribution can be a good bellwether for underlying community health. That's because this process touches elements of community cohesion, governance and contribution in a way that many other individual metrics cannot. However, this must be balanced against a natural shift toward fewer contributions as a project reaches maturity.

What's next, and how can I help?

When it comes to determining metrics for sustainability, the CHAOSS group continue to explore and document metrics across a number of lenses: diversity, value, risk, evolution and 'common' areas of focus. Working groups maintain a contribution process[4] for each area.

The metrics published on CHAOSS' site at https://chaoss.community/metrics are released on a six month cadance. For more information on getting involved see https://chaoss.community/participate/.

[4] https://chaoss.community/metrics

Engaging with and contributing to open source communities

The Challenge:

If you work for a company, there's a chance that you'll come across some functionality or process that relies on open source projects to work. That means that a lot of people are interacting with open source as part of their job.

Sometimes, that results in issues. Because, of course, open source projects are often maintained differently to subscription or commercial products. Even if a person has the best of intentions, it can be tricky to navigate discussions around fixing, improving and amending.

If you're engaging with open source communities, how should you go about it? And if you're contributing, is there an etiquette or process you should be aware of?

The Bad...

So what does it mean to be a good, accountable member of the open source community, if you're also a corporate?

Sometimes, you get a louder response when you ask what you **shouldn't** do. Discussions like these - including the one that took place at last year's Sustain event - tend to throw up similar complaints, including:

- "Buying" maintainers. This could be outright (through practices like acquihires or directly recruiting with the intent to pull them off a project), or subtle (for instance, by only offering financial rewards for specific features)
- Strip mining projects
- Consumption without contribution. This doesn't just pertain to using the code directly, but also to more negative contribution patterns such as opening issues or demanding features without contributing back
- Attaching strings to contributions
- Risk-averse legal department contribution policies

- Copyleft violations
- Not even being aware of what they're using, or willfully dismissing this information as irrelevant.

It’s often hard to judge definitively, but users vary on a spectrum between supporters and profiteers.

In the FOSS ecosystem, there is a lot of for-profit behaviour, largely because most open source starts from a "scratch your own itch" mentality, and everyone who contributes to open source does so out of self-interest. When this self-interest doesn't involve a culture of giving back, it becomes problematic.

...And the good

That said, no one improves without some idea of what they can do better, and how. So, after looking at how companies can misbehave, Sustain also discussed what good corporate open source members do, and what accountability actually looks like.

So what do good companies do?

- Push code upstream by contributing back to open source projects, and support any forks or branches which they've made to improve upon the original code, independently, if the original project goes a different way
- Support their dependencies
- Have clear contribution policies designed to encourage employees to contribute and to stay active as open source maintainers
- Maintain transparency about their contributions
- Hold employee behavior to a standard. They don't just slap a Code of Conduct on a repository for open source participants, but also apply the code internally and hold employees accountable
- Work together with the community early, not late
- Exit gracefully with succession planning
- Stay public and fair in governance

- Clean up after themselves. They know how to document their code, close open threads, and transparently communicate their roadmaps and changes to their communities
- Measure value captured, and give back accordingly
- Participate actively in their communities with both code and funds
- Archive or delete dead projects
- Report out on any success/failure to achieve their goals
- Determine the value they derive from FOSS projects, and publish a contribution goal that matches it
- Act transparently. Transparency has to include a lack of hidden agendas, clear goals, public execution of work, and a public dedication to diversity.

Monitoring, and speaking out

Once you've communicated some of these best practices to organizations, how do you ensure people stick to them?

Some of the schemes of accountability discussed include:

- Certification badges (for instance, membership in the OSI or Eclipse)
- A clear pathway for complaints and suggestions internally, amongst current and past employees
- A documented mission statement/vision for community that aligns to true business goals
- Labor organizing

As an employee, there are steps you can take to help your company be a good citizen of open source:

- Call out bad behaviour, reward good behaviour
- Score your employer

- Actively court corporations that do the kind of work you'd like to be a part of, or reach out to them for partnership and collaboration
- Communicate suggestions and offer "pressure"
- Open conversation in the community of the OS project
- Start conversations about financial expectations. This involves not just measuring value, but also staying realistic while encouraging active contributions.
- Accurately report percentage of contribution and diversity metrics
- Implement consistent peer-reviewed methodologies. Use inner source to enable open source.
- Aim for community participation in decision-making. Invite in the community early and often.
- Fund an accountability officer to report out transparently on success/failure of corporate involvement

All of these points were discussed in greater depth in the Corporate Accountability and Authentic Participation workshops that took place after Sustain. That work, led by Justin W. Flory and Duane O'Brien, came out with recommendations for corporations here. Discussion continues on the Sustain forum.

Getting past participating blockers

Code is normally open sourced so that *anyone* can participate in its creation. For community members who can find the code and understand it, participating isn't normally an issue.

However, for large organizations, who are increasingly important to the sustainability and creation of open source code, there are often blockers to participation. These include a lack of awareness, contribution culture, and friction in contributing.

Lack of awareness is multifaceted: often, developers at a company may work in open source, but they have difficulty explaining this work, and the ROI to their managers or to the executives. On the other hand, the C-suite often doesn't know how useful open source can be to a company's mission, and either aren't aware of how open source works, or of how to support it in their company.

This is also part of the contribution culture problem: open source developers have built a culture of giving back code and energy, in the spirit of enlightened self-interest.

You help me, I'll help you, and everyone benefits from working in the open.

However, many “closed-source-first” thinkers may not be used to the culture of giving time and money towards open source contributors, and there can be a breakdown in communication and contributions. This friction can be exacerbated by a lack of support cultures within the businesses: for instance, legal teams or patent offices are frequent offenders that can slow down or hamper open source work, by not understanding the concepts or by applying their own methods onto open source culture.

And, of course, these are only three barriers: others could be:

- language or geographical barriers in a community
- the lack of clear pathways to gain influence for individuals or businesses
- the lack of time or resources
- the need for internal open source champions to keep things going
- the need for marketing for open source projects
- ...and so on.

Ultimately, for open source to work in the enterprise, these friction points need to be addressed. A Sustain working group met last year to focus on some strategies to help this process.

Use Innersource

Innersource is the means by which you use open source processes internally at the company. So implement innersource processes to mirror open source methods. Instead of a meeting, send a pull request.

Innersource can also be used to train developers into a tit-for-tat system of work, as well as a culture of giving back and sharing with the wider community. Documentation internally is also crucially important, as code's longevity has a direct impact on a company's bottom line.

Talk to Legal

Train legal teams about open source licenses, and work on compliance best practices. Implement fast tracks for small asks - spending a week checking the license for a dependency isn't useful.

There are tools that can help with license compliance - Fossa, or Clearly Defined - but they aren't a replacement for talking to the legal team and setting up clear pathways towards open source usage. Implementing legal templates for devs to retain copyright while working internally would also help, and limit IP exposure for the company in the long run.

Map out how OSS is used

Companies often lack buy-in not because open source isn't part of their process, but because they don't understand how it furthers their goals. It's up to code-adjacent employees to map this out.

How OSS is used differs for each company; whether it's building out the core of the business model for large tech firms, or being part of the service delivery that furthers the mission of a non-profit.

Looking at the actual use of code written or contributed to internally will be crucial to helping change culture at a company. Lay out how OSS can help operational needs; what underlying protocols or tech the business depends on (for more on this, check out Back Your Stack); what products or services are built on OSS; and how OSS can be used for branding, recruiting, and retaining great developers.

Talk about giving back

Open source isn't altruistic; coders commit to projects because they believe that it will help them to do so. But this doesn't mean that it is selfish. The community and ethos depends upon a willingness to cooperate to build great resources together.

There are benefits to buying into this process. Talk internally at your company about the exposure that individual developers, teams, projects, and the company as a whole will get by giving back. Bring up conversations about the opportunity to influence features, and future development of independent projects, while ensuring

that they are diverse and sustainable on their own in the long run. Explore ways in which the company can be a good OSS citizen, particularly involving funding, whether that means a percentage of developers' time given each week to open source, or contributing to the financial growth of the projects.

Ultimately, the approach taken by enterprise towards open source influences the impact that they will have on open source projects, and on the perception of the company going forward.

As the movement grows, it is becoming clearer that transparency is the most important part of these efforts. It's important for all contributors to have shared values, personal ethics, and a common mission for a community to grow. By stating these publicly, and working together with diverse individuals openly and honestly, the stigma of corporate influence can be ameliorated.

Enterprise players in the space also have a large role to play in sustaining projects and the open source community as a whole, by making a space for contributors to work full-time on open source.

It's often the enterprise who funds events, building local developer scenes, providing inspiration for new developers. and helping the ecosystem as a whole share its work. By acknowledging the important role that large businesses can play, and by pointing to successful examples of open source companies, blockers to participation can be slowly broken down and removed.

Accountability and transparency goals

So - if we're serious about building an empowering and collaborative working environment between corporate users and open source - how do we agree on what we expect from that relationship?

In previous years, there's been extensive discussion on what it means to be a corporate member of an open source community, and how to participate transparently *and* authentically.

At last year's Sustain event, the discussions centered around these different aspects of authenticity:

- **Maintainer responsibility**: Open-sourcing your own projects or participating in existing communities. What is the responsibility of the maintainer?

- **Guiding hand**: Navigating contribution intent with poorly-communicated agendas. How can large organizations participate without taking over the space a project operates in?
- **Participation**: Making key decisions while involving a community. Who gets to have a voice in decision-making, and what does that look like?
- **Equalizing knowledge**: Understated value in documentation.
- **Undervaluation of open source**: Communicating less obvious perks for engaging authentically.
- **You and corporation**: Relationship between individual contributors and the organization they represent in their contributions

Largely, much of this work continued in follow-up sessions after the event.

Defining baseline Principles

Soon, it became clear that a core set of principles were needed as metalanguage to conceptualize authentic participation. These Principles are a starting place for future work, such as a best practices document, a certification body, or other mediums.

1. **Authentic Participation Starts Early**

   If an organization shows up with mature, fully baked contributions over which the community had no input, it's hard to make them feel invested.

2. **Authentic Participation Puts The Community First**

   When an organization and the community want different things, the community needs to come first.

3. **Authentic Participation Starts With Listening**

   Are participants showing up to projects with no historical context and telling them everything they were doing wrong? Or is everyone trying to learn the other's perspective?

4. **Authentic Participation Has Transparent Motivations**

Without a shared understanding of the motivations, it's impossible to resolve differences of opinion effectively.

5. **Authentic Participation Enforces Respectful Behavior**

   Participants agree to adhere to community-established codes of conduct. Organizations commit to holding their participants accountable for their behavior.

6. **Authentic Participation Ends Gracefully**

   No sudden withdrawal of resources without notification and an exit plan. Clear documentation also allows the community to pick up projects when a company decides to withdraw support.

What's next, and how can I help?

There are many different ways that we can build a closer and more collaborative relationship between open source projects and corporate users. Part of this could involve participating in some of the group listed above, which are aiming to develop a shared consensus around behaviour, and giving back.

But the work also starts in the organizations themselves. Maybe you would be interested in becoming a "Champion" for open source in your workplace; explaining the value of engaging with communities, and the best practices for doing so? And if so, maybe your story - and where it worked, and where it didn't - might bring valuable experience and context to the discussions taking place on the Sustain forum in future?

Dependency Tree funding

The Challenge:

There have been several projects that focus on distributing funds to maintainers and communities. The most popular examples are OpenCollective and GitHub Sponsors. These platforms are inspired by projects like Kickstarter, Patreon or GoFundMe, although they are much more tech-specific (or friendly, as is the case with OpenCollective, which also allows other sorts of collectives).

These platforms and others provide a vital service? So how they are different, and what their shared challenges and benefits are for the ecosystem at large?

What's out there?

There are a number of options for those seeking funding. Some are developed, and some more nascent. So which are most appropriate for your projects?

OpenCollective

OpenCollective allows individuals or groups to easily set up a "collective", which can start raising money without having to go through legislative processes (such as founding a company, registering for tax returns, assigning shareholders). But you can only fund individual collectives.

This is one of the reasons it incubated BackYourStack; as a way of funding dependencies all at the same time. This project is mentioned here because it is possible to fund a suite of projects through OpenCollective if there's a single collective; a good example is the Unified ecosystem, which has a single collective for hundreds of related repositories.

GitHub Sponsors

GitHub Sponsors focuses on people and projects that enable funding. This is similar to OpenCollective, and you can even allocate money from GitHub Sponsors to OpenCollective using a custom .yaml file. At the moment, it's not possible to fund a dependency tree, and the success metrics are somewhat opaque. GitHub tries to judge whether or not a maintainer is able to pay their rent with the money that comes in, as a small litmus test for success. Like OpenCollective, though,

large, charismatic projects and people tend to get more money than smaller, less public projects.

Patreon

One of several patronage-style platforms, this funds people directly. Still, this is used by several people to fund multiple types of projects - for instance, Feross Aboukhadijeh uses Patreon to fund not only StandardJS, but also WebTorrent, both of which he maintains.

Kickstarter and GoFundMe

Two of the most popular crowdfunding platforms, these are useful for planning specific projects and getting buy-in from customers or users before actually building them. FortAwesome used Kickstarter to fund one of the most successful Kickstarter campaigns, which allowed them to work on their suite of tools knowing that it was sustainable. However, as above, this only works for specific projects, and the investment time in getting a successful campaign is prohibitive for most projects.

Gratipay (no longer running)

Gratipay (née Gittip) was a platform that allowed people to tip individual donors. Much of the work from Gratipay went into understanding how donation models are viable in the real world, and the founder, Chad Whitacre, was instrumental in helping to run the first Sustain.

All of these projects, as mentioned, lack the ability to fund many projects at once (at the moment). So, what projects have overcome this?

Back Your Stack

BYS works by allowing people to allocate funds to an entire suite of projects. The website imports your dependency manifests, and then selects projects which already have collectives set up on OpenCollective. In the future, they hope to also check GitHub Sponsors and other donation models. If a project doesn't have a way to receive money, then the funds are held in abeyance on OpenCollective until a maintainer comes to turn on their collective, at which point they receive the money available.

FLOSSbank

Flossbank, started by engineers at Google and Amazon in their space time, is a project that also gives money down the dependency tree. One of the main differences between them and BYS is that it doesn't hold the funds in abeyance; rather, any amount of money that goes through the system goes to all available projects, and is split equally between them. So, if maintainers want to get money for their projects, they need to sign up, which will then allow them to receive a share of the total funds.

It's unclear whether this actively disincentivizes participation in terms of developers engaging more developers to sign up. Part of the model involves developing a badge to show that a company is sustainably giving back to their communities, which can help drive adoption. Additionally to the donation model, it also has a novel wrapper for the terminal that allows developers to see ads when deps are installed. The lion's share of ad sales from the platform, which uses Ethical Ads created by ReadTheDocs, goes to project maintainers; some is kept with FLOSSbank to keep the project lights on.

FairOSS

FairOSS also uses a donation model. The difference is that it doesn't just rely on cash donations; instead, it actively goes to companies that use open source and asks them to hand over equity or other services to open source projects they depend on. That way, when the company IPOs or is acquired, some of the windfall goes back to the open source projects that the company was built on. Badges and goodwill incentives help companies justify this work. FairOSS came out of a successful Python consultancy, and is currently being incubated as a potential means to grow the ecosystem in the long term, not in the short-term.

Tidelift

Famous for being one of the first "fund this entire ecosystem" projects, Tidelift effectively tries to bundle together entire dependency trees to sell support and maintenance for them to large corporations who aren't able to easily use donation platforms due to discretionary spending limits. By bundling many packages together, it claims to be able to reduce liability for the enterprises which buy their software. Some of that money trickles down to maintainers, who sign a contract with Tidelife involving support expectations, mostly in the case of security incidents. With substantial VC money which needs to be recouped at some point,

Tidelift has been slowly building their offerings and reaching out to more maintainers.

Shared Challenges

Other platforms do exist. Focusing on the last four mentioned, each of these different dependency-funding models has various levels of success and adoption. All of them share similar problems, however:

Mapping the software dependencies

All of these projects depend upon the concept of mapping the dependency graphtree: the tree-like structure that is created as developers include the code of others in their own. The software you depend upon - and the software it depends on - creates a complex and often inscrutable web of interdependency that *should* be supported en masse if we wish to create a sustainable ecosystem of software. This approach is not without its own issues. Dependencies are easily missed. And the institutions that support these projects (TC39, The Python Foundation, Ruby Together, NumFocus etc) are themselves a web of organisational dependencies.

Distribution of finances

Distributing financial contributions to an interconnected ecosystem of projects means solving two problems: allocating funds and distributing funds. Distribution is becoming an increasingly trivial problem due to the works of those mentioned above. However, allocation of funds asks us philosophical questions about the value of another's work and leads us straight to a number of difficult conversations.

Factors to be considered here include usage, scale, complexity, effort and value gained. All that can be said today is that no one solution has emerged as being definitive, and that we are not having these difficult conversations in forums public enough to make progress.

Holding funds in escrow

Only 0.4% of open source projects ask for money. For the rest, someone needs to either contact the maintainers (which takes effort), or institute a method for storing funds until the maintainers claim it, or reallocating those funds elsewhere in the meantime. Other options may exist, but for now this problem defines the differences between several of the projects listed above.

Encouraging donations

Donations are, by definition, not enforced. If they were, it'd be a fee or a tax. So, how does one ask for donations effectively? How do we encourage companies to give what they can, or to give back? Is money the only possible way to show support for the OSS ecosystem? How do donations fit into standard enterprise budgets? Is the donation model only devs giving other devs the same money? These are open questions that need to be approached, if not answered, by each donation-based product.

Funding the platforms themselves

And, finally, all of this work takes work. How do we ensure that this work is rewarded? Some projects, like Flossbank or BackYourStack, are run by volunteers. Others are supported by enterprise models. Funding sustainability work sustainably remains one of the harder problems in this space.[5]

What's next, and how can I help?

This space is still nascent. There are some ways of framing the entire donation-to-dependencies model which can help inform this work.

Some of the discussions that have already taken place have thrown up some potential models:

Quadratic funding, or alternative payout schemes

Quadratic funding provides a democratic mathematically optimal way of distributing matching funds to projects based on community votes (and assuming that these are optimal themselves). FairOSS's purveyance of equity to maintainers is a similarly novel way of allocating funds. Other methods would also be interesting to explore, such as payouts for projects which are depended upon by the most amount of projects, either in a single dependency tree or across dependency trees.

[5] To retroactively fund the person who wrote this paragraph, consider giving to Sustain's OpenCollective.

Shared resource allocation between projects

When funds are given to a single project, the governing body for that project decides where the money goes, and that's that. For larger donations which fund across different nodes or leaves in the tree, it's possible to earmark money that would best be spent across multiple projects simultaneously.

For instance, hiring a community manager for multiple projects may be in each project's best interest, as individually they couldn't afford the expense. Likewise, funding business development work for multiple projects may ultimately help each of them build sustainable models which don't require tenuous or infrequent donations

Metric-informed allocations

How about allocating donated funds differently for projects which have clear, good metrics of community health? This is the part of what is successful about the grant funding model: the grantors spend time asking questions for each grant, and ensuring that their money will be well spent. This is not currently something that most of the donating platforms check automatically.

At 2020's Sustain event, participants asked various guiding questions which are relevant for all of these models:

- How do we prioritize people over metrics?
- How can we most effectively support people who have contributed to code which we use?
- What is the best way to encourage adoption of payback-schemes?
- How can we ethically use open source code?
- How do we give back to the communities whose code we use?

There currently isn't a Funding Platforms WG. But the BackYourStack project in particular is always looking for more devs, and some of the people involved in

this space have been guests on the Sustain Podcast. To get involved, the best place to reach out is probably the Sustain forum

Academia and open source

The Challenge:

What makes academic or specialized open source projects different from the traditional web or operating systems OSS projects. Academic projects don't have the same constraints or goals, and they aren't funded in the same way. As a result, sustainability for these projects differs significantly. And there is no clear definition of what sustainability would mean for these projects.

How different?

At Sustain 2020, there were a number of discussions around this. For instance, not all projects need to persist with active development. An undergraduate's thesis work that's put on GitHub is unlikely to need to be used by anyone else.

In the same vein, work published with an academic paper published on PLoS may only be used by a few researchers to verify the claims or to make similar ones, but won't be put into serious, enterprise-ready use.

For those that do, the idea of using intra-project collaboration as a means towards sustainability was suggested. Specifically, collaboration could be worked in as part of proposals (if it isn't already), and government funding should require it. This could also be used to foster connections between academia and industry.

What's more, most academic projects lack community managers or developer relations. Academic developers may not be able to market their projects the same way, and there is a different incentive structure for doing so, especially as code often isn't used as a metric for academic job qualifications, unless attached to a body of active research with publications.

Another factor is that academic projects often don't have a wider structure which is able to sustain them, or advise on how to translate their work to other areas, how to understand, implement, and enforce licensing, and how to run an open source community.

What's next, and how can I help?

There are many projects working to change this. For instance: URSSI.us (the American Research Software Sustainability Initiative); codeforscience.org; carpentries.org; CHAOSS; the Research Software Alliance; and others. There are also nascent open source program offices at some universities, like RIT and Johns Hopkins University.

There is a good deal of work to be done in this space. For instance, the working group noted that it would be good to write up best practices for university administrations to facilitate OSS development. A collection of OSS success stories would also help normalize the idea that open source can lead to wider impact for everyone involved. A vision document would be useful, and a checklist on how to start successful academic software packages.

Much of this work is being done through the OSPO++ program, run by Jacob Green of MossLabs. Its goal is to build 10 university OSPOs that will connect and collaborate with each from the get-go, allowing for cross-pollination of ideas, funding, and students.

The working group itself is ongoing. For more information, get in touch on the Sustain Forum, or email Richard Littauer at richard@burntfen.com for details.

Evolving open source software

Updating the Open Source Definition

The Challenge:

In 2018, Heather Meeker (et al) lit the touchpaper with the Commons Clause: a licence 'rider' designed to prohibit the wholesale takeover of open source applications and services by hosting providers like AWS.

MongoDB's Server-Side Public Licence (SSPL), Licence Zero, The Hippocratic Licence and similar efforts - often collectively referred to as 'ethical licences' - have attracted a significant amount of debate among stakeholders in the open source community, each with their own interests and values.

How to progress

So what are the problems in this area, and how can we - as a community - resolve and advance them?

- There's potentially a lack of understanding around the "why" of open source which could lead to a generation of developers circling the same problems. Perhaps we need to ensure that those who learn how to develop software are educated as to why licences matter and what the benefits are of having one for projects?

- There appears to be a chasm between "which licence should I choose" and the ramifications of that licence. Could we bridge this gap by providing examples, or asking someone whether they want to be more like project X or project Y? Choose a license https://choosealicense.com/ (maintained by GitHub) could be modified to bring it up to date and more outcome-and-example driven?

- Platforms that host *source available* software are often not as forward as they could be about the lack of a licence. Does this cause us to miss an opportunity to drive open source adoption and encourage participation? Should platforms like GitHub provide simple, one-click prompts to ask maintainers to state a licence?

- Projects like ClearlyDefined are working to clarify inaccurate, incorrect or ambiguous licence declarations on public projects. Platforms should establish a process for picking up these amendments and recategorising projects accordingly.

A revised definition?

Over the last three years The Open Source Initiative (OSI) has drawn a degree of negative comment, due in no small part to its licence review process and its steadfast protection of the Open Source Definition (OSD). As a result, there have been a number of ideas centred around unpacking and addressing some of the issues:

- The OSI is a charity with an elected leadership, but we often see critics describing the OSI like a self-appointed authoritarian organisation. It s*eems* like this would be easily countered with better communication.

- It is difficult to appreciate some of the narrative around the decisions made during license reviews. The OSI could and probably should do more to explain the context around decisions on licences submitted for review, particularly rejections.

- There is a similar lack of narrative around the reason the open source definition was created and what it aims to achieve. Put another way: what norms does the OSD enshrine and by extension what values do the OSI seek to maintain?

The open source definition (OSD) is the yardstick by which new licences are judged when submitted for approval by the OSI. But some have argued that this isn't the only set of values that *appear* to be referenced when assessing new licences. Discussion and decisions sometimes reference previous determinations, drawing parallels to case law. And while the goals of the OSI with respect to the approval process[6] and the licence proliferation committee[7] are documented, it is often the nuances of previous determinations that affect the outcome of a review.

What's next, and how can I help?

So is there something to be said for evolving the open source definition or, at least, having a process for evolving it as licences are reviewed?

This might enable the OSI to more clearly express the values it upholds and the intent of reviewers when reviewing licences. It may also provide a process to

[6] https://opensource.org/approval
[7] https://opensource.org/proliferation-report

clarify the definition to improve later submissions. At the same time such a process may combat some of the criticisms of the OSI by providing a mechanism for stakeholders to
evolve the definition, and by extension the review process, as an open and inclusive community.

In previous discussions at Sustain events, there's been some agreement that a process for adding annotations and clarifications may help test an approach in a forward manner, and that this would allow iteration and build confidence at smaller scales. A move toward constitutional processes may also work due to the vast scale and board stakeholdership within the community.

Any process should also:

- Identify stakeholders and contributors
- Be transparent and traceable — to which many pointed out that the current mailing lists are not easily navigated or searched.
- Protect against being forwarded by those with the most time — which often compounds privilege
- Support 'summary statements' that provide an overview of the factors that influenced a decision
- Protect against bad actors both submitting into and contributing to processes,
- Support extremely large audiences, stakeholders and affiliates — with a note that this could be used as a strength, perhaps requiring a minimum number of supporters to progress to later stages.
- Support regular review timelines and retrospective impacts as the result of a previous decision if updating a definition

There has been interest in creating a working group within the OSI to further this idea, if the committee would be willing to review conclusions and recommendations.

Ethics

The Challenge:

The ethics of open source is an increasingly impactful, and popular topic.

The majority of tech culture has been willing to let ethics sit as a side dish while money was on the table. However, recent events have forced a more urgent discussion.

For instance, consider the influence of social media on elections globally; the weaponization of open source for national security, in particular involving fraught discussions involving ICE and a Chef module; or the influence of open source code in helping the public understand the spread and urgency of Coronavirus.

Ethics is now “mainstream”, to the point of congressional hearings televised on international news. But what does ethics in open source mean?

This topic has been covered elsewhere. The Ethical Source movement has started to take stabs at using ethical licensing to solve issues inherent in open source by adopting better licensing practises. But last year’s event was the first time it was covered at Sustain, and the conversation varied widely, starting with defining scope, considering challenges, strategizing on how to make change, and applying what's next.

Potential roadblocks

Reminding developers to act ethically is important. But so is appreciating difficulties that make it trickier. These include:

Licensing

The Open Source movement depends upon licensing code freely for anyone to consume. A developer can either use a limiting license, or open source it - but, under most definitions of open source, they can't decide how their code will be used by another developer.

This is enshrined in the OSI's definition of open source, which prohibits alternative licenses which limit action, most famously for licenses in the GPL family. Ethical licenses, such as the Hippocratic License or the edited MIT No-Evil license, are therefore not FOSS.

Money

Another aspect of open source that is difficult, ethically, is money. Unethical software gets more money, in diverse ways. On the one hand, proprietary code can be sold and make money on the code itself, whereas FOSS can't. If one takes it as a given that open source can sustain itself through practices like open core models, it still follows that proprietary code is necessary for making money.

Beyond this, code which enables Palantir, drones, or gambling almost certainly makes more money than the majority of open source libraries, due to the nature of their ecosystems.

Finally, money, as a unit of power, is ethically fraught in itself. Exploitation is lucrative, and it's expensive to be intentional about not exploiting others. On another more ideological note, making money is often seen as a moral good, if not an imperative, in capitalist societies. So - what does being an ethical open source developer who needs capital mean?

Enforcement

Often, these sorts of questions lead to a sense of powerlessness on the part of the developer. Developers don't have a say in what they build if they're working for a corporation with different ethical goals; the buck often stops with the paycheck.

This sense of powerlessness leads to the path of least resistance; it's easier to ignore the ethical ramifications of an algorithm. Of course, caring about ethical ramifications is often a privilege of those working at higher levels of abstraction; it's hard to argue for the ethical ramifications of each module in a dependency stack. It's also difficult to know where to even raise these sorts of questions, as discussion of ethical considerations of work isn't normalized in the tech industry. And, while individual action is permitted when it comes to creative genius, it is disincentivized when it comes up against organizations, communities or society at large.

Who enforces ethics for the tech industry, anyway? Certainly not congressional oversight committees, nor transnational organizations like the UN or the Hague. Multi-jurisdictional usage leads to a defraying of any singular coder's ability to bring up ethics as a problem to be solved in their work. In short, ethical dissent is

difficult, to the point that it can be hard to internalize that it is worth thinking about.

Strategies for enacting change

Okay, so we've talked about the hurdles. How do we start talking about change?

Labor Organizing

It's an unpopular topic with companies, as seen by harsh steps taken by major players such as FAANG to stop their employees organizing.

But labor organizing is one of the most effective ways for coders to argue in favor of ethical usage of their code. It doesn't necessitate socialistically grabbing the means of production, either; instead, a wide statement of ethical intent can be adopted by developers and used as a bargaining chip to influence the direction of code.

There's much to be gained in labor organizing, such as showing an example for how employees should self-regulate. Ethical organizing can also happen at the community level. For instance, part of the Io.js fork in Node was based on a desire for better ownership of the community and enforcement of the CoC. Forking itself can be a political, ethical act.

The first step is to express interest in labor organizing, often by talking internally, or looking at US unions that have tech (such as the AFLCIO, or AFSCME). There are various guides online on how to organize, which is a difficult topic at the best of times. A non-America-centric view can also be helpful here, as examples: for instance, England & Australia have broadcast workers and communications workers.

Ethics Boards

Building an ethics board can be another way to work ethically. There are frameworks, standards, and experts from other fields such as civics, health, and institutional review boards in education that focus exclusively on this topic. These could be all applied to open source in some way. While arguments for getting companies to hire, discuss, and ultimately implement Codes of Ethics are still unclear, this would be a promising avenue.

Stay informed

Tech doesn't exist in a vacuum. Staying informed about internal law and rights can be useful for knowing whether or not code will be used in an unethical way. Also, staying abreast of current developments in tech can be useful. For instance, GDPR was a major milestone in respecting user privacy, as is the European discussion around the “right to be forgotten”.

Share stories

While seemingly a bit trite, simply sharing stories of ethical difficulties can be incredibly useful. Not only does this often offload some of the mental weight of an issue, but it also normalizes talking about ethics with other developers, and models how to approach ethical issues in the future. Holding a conversation over coffee, starting a private email group, or organizing a session at a conference is an incredibly effective way to find allies who are interested in breaking down a problem.

Invite diverse perspectives

One of the most effective ways to learn more about how you or your company relates to ethical behaviour is to talk with someone who sees this differently, has a different background, or, better yet, disagrees with you.

To make these conversations happen, lower the barriers for contributions in your projects, which will invite a wider group of users. Invest in diversity, on as many vectors as possible. Build tooling to make it easier for users to understand the code you think has ethical implications, and educate your users with good documentation. Add this tooling and educational materials to your onboarding material, both inside and outside of your projects. If you're at a company, bring in HR when possible to help with this.

There are now online resources that help with this. UC Santa Clara and RIIT both have online creative commons licensed courses to teach ethics with code.

And, finally, institute no harassment policies and codes of conduct. These are clear signs that you take ethics seriously, as discrimination and harassment are almost always unethical by most standards.

What's next, and how can I help?

The ethics working group has been meeting actively since Sustain.

At the 2020 event, some future avenues that were listed included building an ethics board, facilitating labor organizing for open source works, and working to normalize ethics as part of development training.

Since many of these topics can be controversial, this work has largely been done in private groups or by individuals. The working group has also decided to team up with Ethical Source, a community of open source developers who created an ethical license, to start a Podcast on ethics issues. Tune in, join the WG by posting on the Sustain forum, and get involved.

What new communities look like: Open Source in Africa

The Challenge:

At its heart, open source is a global community, built on shared ideals. But the challenges that face maintainers and other community members can be very different. This can be down to a number of factors, including geography. To understand what open source could be - and to collaborate more effectively - it's worth seeing it through many different eyes.

For example, let's consider Africa.

At the last Sustain event, there were four members from Nigeria, all involved with Open Source Community Africa (OSCA). OSCA is a community organization for open source lovers, enthusiasts, advocates and experts within and across Africa. Its aim is to increase the rate of credible contributions to local and global open source projects, by African software developers, designers, writers and others involved in technology.

They aim to change the perception of Africans from "just a billion users" to "the next billion creators". So what can we understand about Africa's community, and what opportunities are there to expand?

Key aspects of open source in Africa:

Since the meeting in Brussels, there has been another conference in Lagos that also focused on community organizing in Africa. Over a hundred participants gathered to paint a fuller picture of what community engagement in Africa looks like.

At this conference, the conversation focused on four key aspects of open source in Africa: funding, documentation, design, and OSS programs that have influence in the community.

Fund-raising
The African tech scene is severely tempered by underfunding. And, without funding, it's very hard to sustain open source developers or development, either for individuals or in companies.

This problem is partly cyclical. Funders don't know how to fund African development, and so they don't have a presence on the continent, and so they don't fund there. The funds which do support African development often aren’t very large, or they focus on particular communities, like South Africa.

Because of this lack of funding in general, there is only a small or a latent tech scene, which means that open source as a coding methodology is also scarce. As many community organizers are forced to use their own funds or volunteer to keep their OSS communities going, it is difficult to sustainably keep momentum up.

There is some hope. For instance, OSCA was able to fund its conference through personal connections. Where possible, these connections may eventually solidify into meaningful relationships. However, there's a strong feeling that this isn't the "normal" way to fund open source, which makes it doubly difficult to reach out for more funds.

On the other hand, there are some companies that are beginning to make inroads into development here; for instance, Google Summer of Code, or Ushahidi. In time, this may translate into more local companies utilizing open source code.

In future, developing clear strategies for marketing open source in Africa would be fruitful. For now, this mainly involves growing devrel roles in local communities, going to conferences and talking about OSCA.

Documentation

Africa has around one and a quarter billion people. While 700 million speak English, around 7 million of those speak English as their first language.

Considering the vast majority of open source code is written in English, it is hard to understate how crucial good documentation is towards open source in Africa. Documentation needs to be clear, concise, and translatable to have an impact in Africa beyond a handful of anglophones.

However, disregarding the five hundred languages spoken in Nigeria alone, even having clear documentation in English is difficult. There are different dialects of English in Africa, and it is hard to ensure that all of the lexical choices in documentation work for each dialect.

One of the key tasks which can be done by open source contributors is to work hard to ensure that the pathway towards improving the documentation in a project is clear, and that the project is as welcoming as possible. In order to have a better contribution base, you also need to make sure that your document reflects what you want to do. Be clear and concise.

Design

Coding is not the only aspect of open source. One major aspect that has come up repeatedly in OSCA is open source design.

Design is particularly important when considering Africa because of its level of diversity, and because the stakeholders often aren't considered when defining and implementing code projects.

Accessibility, internationalization, and offline-first methods aren't just fancy add-ons when it comes to Africa, but real needs on the ground that need to be met.

Ultimately, design and code are parts of wider ecosystems. Any project that hopes to make an impact in Africa or outside of highly developed nations needs to spend more time working on how someone who has a different approach to tech might access their project. The easiest way to do this is to ask users directly, and to work with local designers from the start to ensure that this happens.

OSS Programs

There are some programs that are already making inroads into working in Africa. A converse of the 90% funding rule mentioned above is this:

90% of developers on the continent have interfaced with GCI, Google Summer of Code, or Google Summer of Docs.

The takeaway from these programs is that it is possible to engage directly with open source projects, and get paid to do so. Getting paid to do open source is rare anywhere, but much rarer in Africa, and just having a public option that shows this is possible is a big step towards normalizing it here.

What's next, and how can I help?

If it wants to be more than a licensing process or a graveyard for abandonware, open source needs to depend upon the communities it serves.

OSCA is the main active community in Africa working on open source. It doesn't just do one thing, but it serves as a catalyst for many types of conversations between developers, doc writers, designers, community managers, and so on. If Open Source in Africa is to thrive, the goal for OSCA is to expand its network and to make it easier for individuals, companies, or OSPOs to work together with communities.

One of the takeaways from Brussels was that it should be easier for funders and engagers to find a way to easily hire and find developers. OSCA is working on this, and welcomes all contributions. From fundraising donations, to strategy, to research - literally anything that companies or individuals are seeking to know about the continent - the best route is to contact info@oscafrica.org.

OSCA covers and has members in the entire continent, across four time zones. There will be a way to collaborate and bond with African developers, as long as coders don't stay in Silicon Valley and stereotype what they think Africa is, but instead get involved on the ground.

If you're interested in helping other open source communities beyond Africa - for instance, in South America, or other parts of the global South - make your networks
as diverse as possible. Allow everyone to join in where you can, and welcome contributions.

You have to make sure that you create a community wherever you are. OSCA is full of people who are not necessarily in tech. Of course, the ones who get hired from Africa often leave, which makes it harder locally - but having people work together is a way to fight "brain-drain".

To be more inclusive, consider projects like Google Summer of Code, or funded project work. Keep the connections alive after these projects end. When building

international FOSS projects, make sure that they are safe spaces for community contributors. Where possible, work with existing local organizers and communities. Use universities to reach into communities. Often, students are able to collaborate the best on code projects, and they have ties to their peers.

Environmentalism

The Challenge:

Open source code is often held up as providing a net benefit for society as a whole. By putting all of our information in the commons, we facilitate technological advancements, making lives better.

However, open source also facilitates technological advances which may not be in the public interest, from military technology to big oil and consumerist fads. The industrial revolution did improve the livelihoods and day-to-day conveniences for billions of people; but it also led to widespread degradation of our environment.

The role of open source, in particular, in the ongoing industrial revolution is unclear. Besides the knock-on effects of widely-adopted code, there are also other considerations, such as the need for new hardware and rare metals, or the electrical demands of data storage or processing power.

Improving the climate impact of open source:

The conversation around climate and open source can be overwhelming.

Individual action is both necessary, and difficult. Employees often don't have influence at their companies. Individuals don't have outsize influence over their local governments; and organizations themselves depend on products and processes that aren't sustainable.

Shaming or using guilt as a motivator for change is largely unhelpful, and structural change will never come easily.

But here are some concrete suggestions for moving the needle a little. Do you have any of your own?

- Switch to a zero-carbon footprint facility for your offices, if possible. Working remotely is even better, if the costs of in-person meetings are low
- Use your developer influence to advocate for better practices in your company
- Institute a “Fridays for the Future” hack at your company; use your free time to work on projects that are helping the environment

- Use open source hardware, as this ultimately has an energy reduction
- Be mindful of data center usage
- Normalize talking about the environment and technology together. This discussion is not mainstream
- Find information on local data centers or local usage of open source data. Ideally, this can be used to lobby towns and municipalities towards best practices. Go up the stack, instead of lobbying from the national or international level down
- Institute a Carbon.md in your projects to talk about the cost of your work
- Build a CO2 explorer to estimate the cost of technological development, just like the AWS cost explorer
- Recycle electronics as much as possible. There are some models where local businesses will learn about what products are easily recyclable or not; this information can then be used to lobby for better hardware. Likewise, working on a sustainable way to stop companies destroying hard drives for the sake of safety would be important
- Remember that individual action is highly limited; we are influenced by our environment. Work on stories that help people change their mindset so that they can work with governments to change attitudes towards tech in regards to the climate
- Institute data demands, such as "You have two years to lessen your carbon footprint, or we're moving to another provider".

What's next, and how can I help?

There is no easy answer to the climate problem. There are only incremental changes which eventually build toward a greener world.

Some suggested projects could include:

- Preparing a list of available resources

- Mapping local companies which have carbon footprints
- Celebrating companies and events that offset their travel costs
- Organizing a “Friday for the Future” in tech, and how can we ensure that developers are able to find projects that will ultimately help the environment
- Building business models to pay for story-telling and lobbying for change for decision makers, either in enterprise or government

If you want to gather support or resources for any of these, or have any other ideas, why not communicate with members in the Sustain forum?

Conclusion

Conclusion

The Sustain Community continues to grow and change.

Through 2020 and 2021, we've existed in an increasingly more remote-first world, and seen various shifts in the open source ecosystem.

Calls for diversity are even more important following the FSF shake-up involving Richard Stahlman. Calls for sponsoring projects directly are more urgent without conference funding to provide for open source foundations. Calls for more equitable funding have been given more fuel through projects like Gitcoin's collaboration with OpenCollective, providing projects with money using Quadratic Funding. The list goes on.

Disagree with anything you've read? Got more to say? Think we ought to have mentioned X? Excellent!

What we've written here is not set in stone. Instead, this can be seen as a leaping-off point for more discussions, more community engagements, and more experimentation with open source sustainability models.

To keep this conversation going, Sustain continues to hold open spaces: our Discourse forum, our working groups, and our podcasts. Tune in, speak up, and reach out.

We look forward to our next Sustain events; currently, we're planning a 2021 digital miniconference, and we're hopeful that one day we'll be able to have regional Sustain conferences again.

Until then, thanks for reading, sharing, and being excellent humans. Keep it up.

Sustain