Upstream First Best Practices Guide

A code review is the primary forum for getting and giving feedback. Patches almost always require iterative feedback/fix cycles. Large patches in particular often require input from many people over a long period before they are ready to be merged.

There are two common mistakes that organizations new to open source make. Here's why they cause problems, and what to do instead:

- Avoid doing purely internal development that is then "thrown over the wall" without discussion with an upstream community, with the expectation that it will be implemented in its entirety. Large patches merged without iterative feedback, especially if developed by people within the same company, will raise red flags and instill a sense of community mistrust for that company that may take a very long time to overcome. Always ensure that large features are planned publicly, with community input.
- Don't limit yourself to maintaining downstream-only changes. Because open source projects are often the consumers of other projects, a downstream-only focus can result in costly refactoring of code to consume new upstream changes. Make sure to maintain changes upstream and keep your downstream (proprietary) work aligned with upstream releases.

Steps to Upstreaming - "I have a great idea for a feature... now what?"

The engagement model for contributing new code may vary slightly from Project to Project. (For clarity, typically, Project refers to the collective platform, such as ODL or ONAP, whereas project refers to the individual development efforts or repositories approved as part of the Project.)

Review the various LFN Project sites to identify where your contribution fits best. Within that Project, Review the list of existing projects/repositories to see where you believe your contributions can be best aligned with what you want to do. You have the best chance at a successful community engagement if you take the following steps:

- Contact the Project Technical Lead (PTL)
- Ask questions
- Pitch your idea to the community
- Listen to the feedback
- Make iterative adjustments
- Seek Approval

Contact the lead Committer or Project Technical Lead (PTL) for the project you're interested in joining

The PTL will have the best knowledge of what that group of developers care about. Introduce yourself and provide an overview of your idea. Your conversation with the PTL will be the beginning of an ongoing dialogue, not a one- shot conversation.

If you are uncomfortable with contacting an unfamiliar PTL directly, contact the LFN Program Manager for the Project and ask her/him to make an introduction. They will be happy to do so.

Ask Questions

- Has something like this been attempted before? If so, what is the status of that effort? Proposing a change without knowing if there was previous history addressing the issue you're interested in is likely to undermine the perceived credibility of your solution, regardless of how good it is.
    - It is possible that there may already be work under way, or something that was previously abandoned for some reason. Proposing a different approach or solution to a problem is something that a community will probably debate, but your approach may ultimately be embraced.
- Does the PTL think this is a good idea / approach? If not, what is missing, or what factors do you need to take into consideration?
- Does the PTL think it would be a good fit in their project? If not, where do they think it would fit better?
    - They may suggest that your work would best be done in another project or perhaps as an entirely new project. If they think it would be best as a new project, ask them for pointers to the "How-To" documentation for proposing a new project, as well as introductions to the people who can guide and support the proposal.
- Are there any hot-buttons or trigger words you should know/use/avoid with this community? Being successful could all come down to how you position something.
- What is the best communications channel to use for initial conversations?
    - Some teams in the same Project may prefer instant messaging over mailing lists. Other teams may be more wiki-focused. There isn't a one-size-fits-all answer, so understanding the preferred communications model is important.
    - Where can you best contribute to the project now, while you are putting your proposal together? Name recognition and trust within the community are much more important in open source than someone's job title or organizational status within your company. Making even small contributions, such as minor bug fixes or documentation improvements, can go a long way to helping you achieve your ultimate goal.

Pitch your idea to the community

- Having gained insight from the PTL and tuned your initial draft, sharing it with the community is the next big step. For most projects this will involve filling out some manner of template that includes the key information about your proposal.
- Never try to bluff your way through your pitch. You are talking to developers, not executives who are several levels removed from the realities of actually making something someone else's "good idea" work. Unlike many internal development organizations, saying "I don't know" or "I had not considered that." are not signs weakness in an open source community. A good response is, "I'd love to hear the community's suggestions on how to address that issue."
- Address any prior history at the outset of your presentation. Point out the ways that your solution is similar and where it differs.
- As a newcomer, establishing that you have been in discussions with the PTL in advance of your pitch will usually garner a lot of goodwill with the community. Never say things like, "I spoke to the PTL and she agrees this is a good idea" unless you have their specific permission to do so.
- Defend without being defensive. Developers respond to more to facts and data more than they respond to feelings and opinions. If someone disagrees in a heated way, don't respond immediately. Someone else in the community will likely come to your defense, even if they themselves disagree with you.

- Initial pitches are rarely accepted “as is”. Expect many questions and possibly some healthy debate among community members. Listen to what is being said and use that to finetune your proposal. As with your initial conversation with the PTL, this is the first step of a longer conversation.

Listen to the Feedback. All of it.

The real power of open source development is that many differing perspectives will generate the best outcomes. This may involve highlighting new functionality that never occurred to you, or uncover a different use case that your proposal cannot address. Take the community’s input to heart because when comes time to seek approval, if there is some key element that you failed to address, someone will bring that up.

Make iterative adjustments

- You are going to get more feedback than you will probably expect. Take that input and keep the dialogue with the entire community running throughout the process. MAKE SURE THAT YOU ARE USING PUBLIC COMMUNICATION CHANNELS TO DO SO. The occasional 1:1 conversation is OK, but in open source the expectation is 100% transparency. This includes individuals within the same company communicating on the open source project.
- Work through architectural modeling and implementation concepts at a high level where key questions can be answered and more importantly the reasons why.
- Avoid the urge to cram too much into your initial proposal for “Release 1”. Think about the long term plan and what a logical roadmap may look like.
- Generate believers. The more you are engaged with the community in defining the scope of your proposal the more trust and credibility you will generate. This is a great help to not only draw in interested contributors, but to gain advocates for when your proposal goes up for approval.
- Identify contributors versus “interested parties”. When a group of developers expresses a keen interest in your project, many people new to open source interpret that as a commitment to contribute to the project, which isn’t always the case. When you see enthusiasm for your proposal, simply ask what resources, if any, they can realistically contribute to the project. Always do a final verification of your contributor commitments immediately prior to your approval request.
- Contributors versus Committers: Most of the project proposal forms ask for a list of Contributors and Committers. Many times when a new group puts their first project proposal forward, Committers are assigned based purely upon a person’s job title within a company, rather than based upon a person’s job role on the project. This practice almost always causes a great deal of pain for the company in the long run, delays the velocity of the project and erodes community trust.
- In most Projects, “Committer” is an actual job role within the Community with a set of responsibilities that are very clearly defined in the Project’s legal charter. While any contributor may make a commit into the branch they have pulled, only a Committer can push that patch back into the main line. A Committer is responsible not only for writing code, but for reviewing the code of other contributors and then either rejecting the patch or executing the code merges into the main code branch.

Seek Approval

This is often the easiest part of the process if you have been diligent and engaged with the community up till this point. In theory, the community should be well aware of the proposal, you should have contributors lined up, and advocates on your side.

After your proposal has been published for the community to review and comment on, you will typically present it to the Technical Steering Committee for your Project. Don’t be surprised if you receive feedback from a different set of people during this period. If your proposal has been effectively vetted with the community, you can usually refer to answers previously provided in a public forum, and be reasonably confident of getting final approval.

Have Questions? Please contact your Project's Technical Program Manager.