# Growing Open Source Seeds

2013

5 min read • 1,133 words

**Themes:** Programming Spiritual

# The Facebook Story

The Facebook Python SDK Years ago, Facebook created a Python library for interfacing with their API. The project had a large number of users (after all, Facebook is pretty popular), but was very rarely updated. One day, the library stopped working completely. Dozens of people trying to build apps and tools on Facebook opened up issues trying to help them resolve the problem. Nothing.

Eventually, the internet got wind of the situation; a very public issue was raised on GitHub that accumulated hundreds of comments asking for an official response from Facebook.

A few weeks later, Facebook responded — by removing the issue tracker completely.

This tragic story is an example of the downfalls of a Public Source Project. Basically, an organization labels a chunk of code that others may find useful as open source. While notably better than the default of keeping everything closed, these projects are almost always abandoned — either via lack of interest, burnout, or change of focus. Motivations are unclear.

# Gittip

On the other side of the fence, we have Gittip, the flourishing generosity platform for sustainable crowd funding.

Chad Whitacre, the creator of the project, strives to make every aspect of Gittip as absolutely open as possible.

- There’s a GitHub issue for everything.
- Major company decisions are voted on by the community.
- Interviews with journalists are live-streamed on YouTube.
- All formal discussions with other organizations are publicly documented and discussed with the community.

> I’m not building Gittip, I’m building the community that’s building Gittip —@whit537

Gittip is an extreme example of a Shared Investment Project. Many large open source projects are run this way — Python, Django, Firefox, jQuery, etc.

All of these projects benefit greatly from the security of shared ownership, extreme transparency, and a high bus factor. New contributors can get involved by following a simple, documented process. This low risk, sustainable approach is an appropriate one — this software is being used by millions of people all over the world.

However, Shared Investment Projects are hardly appropriate for everything — see Tragedy of the Commons.

# Requests: HTTP for Humans

Requests is a surprisingly popular open source project that I created. It's been downloaded 3,300,000 times and I receive dozens upon dozens of emails about it every day. The project is very much developed in the open and there have been thousands of contributions by hundreds of people—embodying the "for Humans" philosophy of prioritizing developer experience.

However, there's one key difference between Requests and Gittip or Python. While I develop Requests in the open and have a thriving community built around it, zero of the project's decisions are made by the community — they are made by me.

Landing somewhere in-between Public Source and Shared Investment projects, the Dictatorship Project features a totalitarian BDFL that owns all aspects of the project. All decisions are 100% made at their discretion. Community feedback is common and highly encouraged, but users should have no expectation of change after providing feedback.

Not only has this approach been successful for Requests, I believe that it is largely responsible for the success of the project. Requests' true value lies in its extreme opinions, which would be lost if decisions were made with others.

There are downsides to this approach, of course.

- The bus factor is extremely low.
- The risk of burnout is extremely high.

- If the BDFL loses sight of their goals, the project is ruined.

# Lessons Learned

I've learned a lot about myself and others while maintaining Requests, including some great ways to combat these potential issues.

## Be Cordial or Be on Your Way

As a contributor, keep all interactions with a maintainer as respectful as possible. They put a significant amount of time and energy into their project, and they don't owe you a moment of their time.

If you're a maintainer, you have the crucial responsibility of being immensely thankful and grateful to any and all contributors. Don't worry about non-constructive feedback. It serves no purpose. Realize that some people just take things way too seriously. Tell them to be on their way.

Be careful with the words you choose. Contributors sometimes take what you say very personally. That oddly-worded and unclear issue that was just opened may be that person's first interaction with an open source developer ever. This happens more often than you think. A little bit of kindness goes a long way. Take the opportunity to educate the user if they aren't asking the right questions.

## Avoiding Burnout

One of the greatest challenges of open source is sustainability. Everyone has a limited amount of time and way too many things to do. Eventually, you'll become the bottleneck for your own project.

Luckily, this can be avoided:

> Open source provides a unique opportunity for the trifecta of purpose, mastery and autonomy. By recognizing the power of these factors, we can keep ourselves motivated and continue to increase our impact. — @geemus

A big part of this is doing less. Meet @lukasaoz and @sigmavirus24, my evil minions. They make my life awesome. When a new issue or pull request comes in to the Requests repository, these gentlemen triage it. This saves me an immense amount of time. Instead of having the constant burden of having hundreds of pending issues and dozens of pending pull requests on Requests, I periodically check the repository and find a nice stack of PRs that have already been reviewed and updated, ready for merging or further comment.

It's been great learning from each other.

> As I fork another of his projects, it occurs to me that I don't program in the Python ecosystem: I program in the @kennethreitz ecosystem.
> — @Lukasaoz

I try to do the same for Flask, Gunicorn, and Werkzeug as well (although I don't contribute nearly as much as I should).

## Learn to Say No

One of the most difficult challenges of maintaining a project is learning how to say one word: **No**.

People are going to ask for features (sometimes crazy ones). They will argue with you and complain that your design doesn't fit their specific corner-case and is therefore totally worthless.

> Simplicity is always better than functionality.— @hintgens

Cater your project to the masses, and you will slowly watch your project transform into something complex, heavy, and difficult to maintain. The best pull requests are the ones that remove lines of code.

The easiest way to accomplish this is to document the project's goals and intensions. When someone sends a crazy pull request, kindly thank them for their contribution, point them to the guidelines in the documentation, and encourage them to send another pull request.

Open source makes the world a better place. Don't make it complicated—principles that evolved into understanding programming as spiritual practice, treating code development as conscious service to human flourishing.