Growing Open Source Seeds

2014

3 min read • 666 words

Themes: Programming Spiritual

Introduction

This talk explores the diverse approaches to open source projects, examining the evolution and challenges within the community. It focuses particularly on the philosophy and sustainability of open source contributions, offering insights drawn from years of maintaining some of Python's most popular packages.

> These insights into open source sustainability predate many current discussions about maintainer burnout and funding, offering practical wisdom from someone who lived these challenges firsthand.

Public vs. Open Source

There's a crucial distinction between **public source** and truly **open source** projects. Public source involves code released with an open source license, but without active maintenance or development. These projects often become abandoned due to lack of interest or maintainer burnout, littering GitHub with good intentions that never found sustainable support.

The **Gittip** project exemplified an extreme approach to open source, embracing shared ownership and radical transparency. Major decisions were made through community involvement on platforms like GitHub, with every discussion and debate happening in public view. This represented a "shared investment" model where contributors actively participated in the project's development, creating true collective ownership rather than mere collaboration.

Dictatorship Projects

The **Requests** library operates as what I call a "dictatorship project" - managed by a Benevolent Dictator For Life (BDFL) who makes all key decisions. While community feedback is encouraged and valued, there's no expectation that it will necessarily influence decisions. This model offers benefits like quick iteration and maintaining a strong, coherent vision, though it carries risks including low bus-factor and high potential for maintainer burnout.

> This frank discussion of the BDFL model was controversial but honest. While community-driven projects have clear benefits, sometimes a single vision is necessary to maintain coherence and prevent feature creep.

Lessons in Open Source

For **contributors**, interactions with maintainers should always be respectful and appreciative of their time and effort. Remember that most maintainers are volunteers, often working on projects in their limited free time. Your feature request or bug report is one of hundreds they might be dealing with.

Maintainers must be thankful to contributors, choose words carefully, and take time to educate users, particularly those new to open source. Every interaction shapes the community culture. A harsh response to a first-time contributor might discourage not just that individual, but others watching the exchange.

Avoiding Burnout

Open source sustainability remains one of the greatest challenges in software development. Projects become difficult to sustain due to time constraints and the risk of maintainers becoming bottlenecks in their own projects. The key lies in maintaining purpose, mastery, and autonomy - the three pillars of intrinsic motivation.

Delegation becomes essential for long-term sustainability. By leveraging a team to handle issues and pull requests, maintainers can focus on larger architectural decisions and strategic direction, improving both the project's health and their own well-being.

The Power of Saying No

Perhaps the most important lesson I've learned is the power of maintaining simplicity by rejecting features or pull requests that add unnecessary complexity. Simple code is easier to maintain, understand, and extend. Complex code leads to technical debt and exponentially higher maintenance burdens.

> This philosophy helped Requests maintain its elegant simplicity despite thousands of feature requests, proving that restraint can be more valuable than addition.

Every feature added is a feature that must be maintained forever. Every configuration option doubles the testing surface. Every abstraction adds cognitive overhead. Sometimes the kindest thing you can do for your users is to say no to their requests, preserving the simplicity that drew them to your project in the first place.

Conclusion

Open source should make the world a better place without adding unnecessary complexity. Success comes from focusing on simplicity, sustainability, and maintaining a clear vision. The goal isn't to accept every contribution or implement every feature request - it's to create something valuable that can be maintained and understood over time.

These principles of ethical open source development evolved into deeper understanding of programming as spiritual practice, treating code creation as conscious service to human flourishing rather than mere technical achievement.