Assessing the Health of Open Source Communities

Kevin Crowston
and James Howison
Syracuse University

Before contributing to a free or open source software project, understand the developers, leaders, and active users behind it.

The computing world lauds many Free/Libre and Open Source Software offerings for both their reliability and features. Successful projects such as the Apache httpd Web server and Linux operating system kernel have made FLOSS a viable option for many commercial organizations.

While FLOSS code is easy to access, understanding the communities that build and support the software can be difficult. Despite accusations from threatened proprietary vendors, few continue to believe that open source programmers are all amateur teenaged hackers working alone in their bedrooms. But neither are they all part of robust, well-known communities like those behind Apache and Linux.

If you, as an IT professional, are going to rely on or recommend FLOSS, or contribute yourself, you should first research the community of developers, leaders, and active users behind the software to decide whether it's healthy and suitable for your needs.

LIFE CYCLE AND MOTIVATIONS

Understanding a project's life cycle and its participants' motivations is useful for understanding why a FLOSS community is important to a project's success.

Eric Raymond claims that successful open source projects usually start in a "cathedral" before heading into the "bazaar" ("The Cathedral and the Bazaar," *First Monday*, vol. 3, no. 3, 1998; www.firstmonday.org/issues/issue3_3/raymond/index.html). Anthony Senyard and Martin Michlmeyr, a former Debian project leader, agree, arguing that a working code base for a successful FLOSS project is usually developed alone or by a very small group before going public ("How to Have a Successful Free Software Project," *Proc. 11th Asia-Pacific Software Eng. Conf.*, IEEE CS Press, 2004, pp. 84-91).

The founders' good ideas expressed in working code facilitate a successful project's second phase: a "creative explosion" in which the product, now public, develops quickly, gathering features and capabilities that in turn attract additional developers and users. What Perl founder Larry Wall calls "learning in public" can be an exhilarating, if difficult, time early in a project's life cycle.

Many researchers have examined FLOSS participants' motivations, but these studies often focused on small samples of atypical projects.

More broad-based research by Rishab A. Ghosh and colleagues ("Free/Libre and Open Source Software: Survey and Study," summary report, Workshop on Advancing the Research Agenda on Free/Open Source Software, Int'l Institute of Infonomics, Univ. of Maastricht, 2002, www.infonomics.nl/FLOSS/report/workshopreport.htm) and Karim Lakhani and Robert G. Wolf ("Why Hackers Do What They Do: Understanding Motivation Efforts in Free/Open Source Software Projects," working paper 4425-03, MIT Sloan School of Management, 2003, http://opensource.mit.edu/papers/lakhaniwolf.pdf) indicates that motivations are quite diverse and include, in decreasing order of relevance,

- intellectual engagement;
- knowledge sharing;
- the product itself; and
- ideology, reputation, and community obligation.

Although reputation is low on the list, its importance does rise with the length of participant involvement. Projects that have an atmosphere of exploration and intellectual engagement, especially early in their life, are most likely to attract the active user community needed for future success. Also important in attracting good developers is a code base that solves a real need.

ONE SHAPE, MANY SIZES

Another important barometer of a FLOSS community's health is its shape and size. Healthy communities are generally onion-shaped, as shown in Figure 1 on the next page, with distinct roles for developers, leaders, and users.

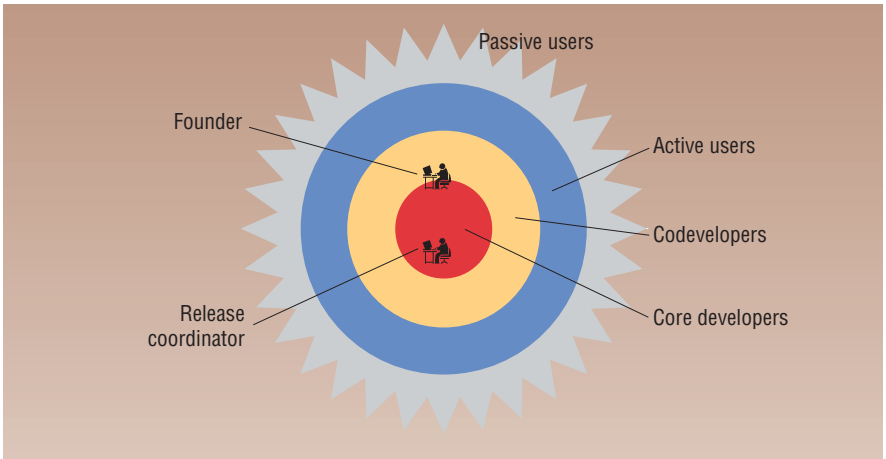

Figure 1. A healthy FLOSS community is onion-shaped, with distinct roles for developers, leaders, and users.

Core developers

At the onion's center are the core developers, those with commit privileges on the source code repository earned through a strong record of contribution to the project. This group can be quite small—three to 10 is adequate for most projects. As every software developer knows, there's a limit to the number of people who can participate intensively before too much time is spent getting in synch.

Our research shows that less than 1 percent of the more than 100,000 projects listed on SourceForge have ever exceeded 10 developers (K. Crowston, J. Howison, and H. Annabi, "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process: Improvement and Practice*, in press, http://floss.syr.edu/publications/crowston2006flossSuccessSPIPpre-print.pdf). Even acknowledging the numerous single-person and dead SourceForge projects, it's clear that projects with hundreds of developers, such as the Linux kernel, Mozilla, and Debian, are the exception rather than the rule.

Project leaders

Also in the core are the project leaders, often the founders. A strong leader or leadership group can carry a project through its turbulent inception to maturity and stability. Contrary to the FLOSS mythology of democratic open project teams, many leaders make decisions without seeking much input. This autocratic style isn't necessarily bad if the decisions have community support, but, taken to an extreme, it can inhibit contributions needed for further growth.

Transitioning project leadership from the founders to others can be quite difficult because, even if they're no longer actually involved, the founders' word still carries inordinate weight. Not surprisingly, change at the center of FLOSS projects is uncommon (J. Howison, K. Inoue, and K. Crowston, "Social Dynamics of Free and Open Source Team Communications," *Proc. 2nd Int'l Conf. Open Source Software*, IFIP, 2006, http://floss.syr.edu/publications/howison_dynamic_sna_intoss_ifip_short.pdf).

It's a positive sign, therefore, if a FLOSS community has successfully managed such a transition. Regardless of who is at the core, however, it's worth asking, could the community recover if the current leadership left? Who in the wings could step forward?

Codevelopers

Surrounding the core developers are the codevelopers, participants who write code but have not yet earned commit access. These individuals typically submit patches for review by core developers before inclusion in the code base and, by displaying interest and ability, can move from the periphery to the core.

This review process enhances code quality by adding extra pairs of eyes, but it can also be time-consuming. Consequently, FLOSS processes emerge that conserve developers' time but seem wasteful of the codevelopers'. What's important is that, beyond an initial creative explosion, development grows steadily over time, and an established social structure emerges that provides a clear vision and has a deep understanding of the code.

Active users

Another sign of a healthy FLOSS community is a wide circle of active users. These persons contribute by testing new releases, posting bug reports, writing documentation, and, most importantly, insulating core developers from a barrage of setup, configuration, and build questions from passive users—those who use the code without contributing themselves.

Active users protect developers from both burnout and possibly frustrated new users. Figure 2 shows a plot of interactions relating to bug fixing in the SquirrelMail project (www.squirrelmail.org), with active users forming a natural buffer between developers and peripheral users. An ideal active user buffer is led by one or two longer-term participants supported by a rotating group of users of varying experience.

DEVELOPMENT PROCESSES

Few if any FLOSS projects are likely to apply for ISO 9000 process certification anytime soon, but that doesn't mean they don't have well-accepted processes. However, these are rarely formally documented, and understanding what Walt Sccachi calls "informalisms" can take time ("Understanding the Requirements for Developing Open Source Software Systems," *IEE Proceedings—Software,* vol. 149, no. 1, 2002, pp. 24-39).

Even very successful open source projects often lack detailed roadmaps, explicit work assignments, or feature request prioritizations. A key aim of making proprietary software processes explicit is to ensure the efficient

use of a fixed pool of resources, but FLOSS projects don't face such fixed pools, either in the number of participants or in the amount of time each one can devote.

Therefore, organizing for fun can be more important than organizing for efficiency. In fact, duplication of effort could be a positive sign that the project can attract resources and is in a position to choose the best contributions. On the other hand, a formalized system for prioritizing security issues clearly benefits some applications. And if the processes are discussed, it's important that these discussions regularly end in action that lets people get back to work rather than in an exhausted stalemate.

Certain essential processes, such as providing and maintaining repositories and download sites, are often both difficult and laborious; these have a high burn-out rate because many participants are driven by intellectual curiosity rather than a service mentality. Such functions should be farmed out to an entity with financial and operational resources, or at least formally rotated among participants. Hosting providers such as SourceForge, Savannah, GForge, and RubyForge do the FLOSS ecosystem a great service in this regard, as do long-term partnerships with commercial ventures.

Managing releases is another onerous task. Hassling volunteers and collaborators to stop being creative and focus on delivering and testing a releasable version isn't particularly enjoyable. Ideally, a healthy FLOSS community recognizes and explicitly addresses this issue, perhaps through a rotating position (like Perl's pumpking) or a time-based release strategy.

In considering a FLOSS project, make it a point to understand the community as you familiarize yourself with the code. Subscribe to and skim mailing lists, find the list archives, and examine the project's Web site. If the project has an Internet relay chat channel, spend some time there—IRC's informality can reveal whether participants actually like one another.

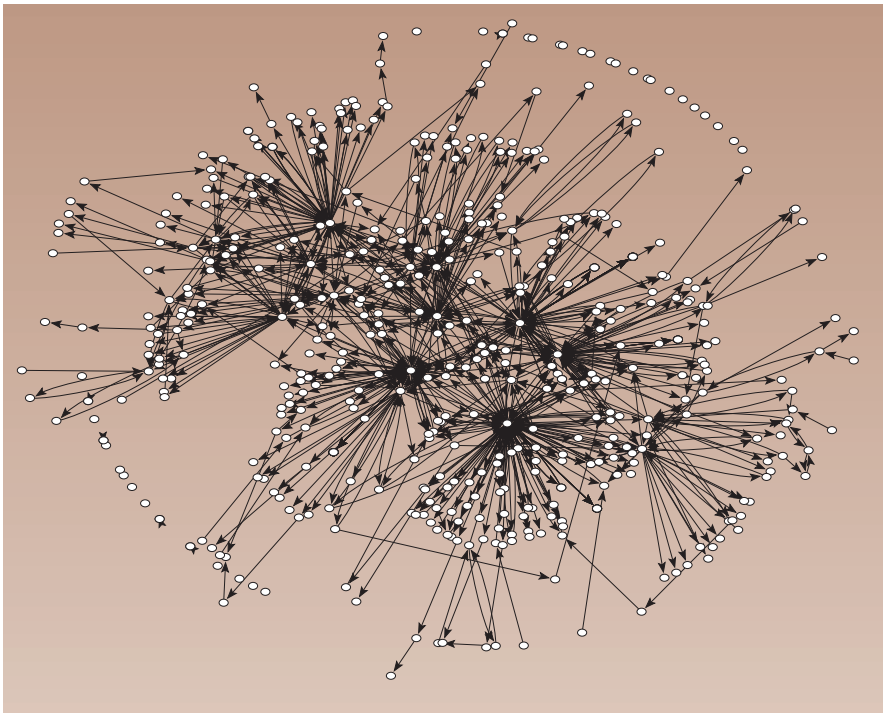

Figure 2. Sociogram for fixing bugs in the SquirrelMail project. Active members—those with multiple interactions—form a buffer between developers and peripheral users.

Many quantitative resources can reveal how a community has evolved over time. For example, FLOSSmole (http://ossmole.sourceforge.net) provides public access to data collected from SourceForge, freshmeat, and RubyForge, as well as tools for graphical analysis of community structure. CVSAnalY (http://cvsanaly.tigris.org) offers utilities and data extracted from many CVS and Subversion projects, such as the number of contributors and their rate of contribution. The Business Readiness Rating project (http://openbrr.org) is developing a more formal methodology to assess FLOSS projects.

If your assessment leaves you feeling that the community isn't right for you, be prepared to consider alternatives, no matter how attractive the code is. Trying to change an existing community is likely to end in frustration and undermine the reasons you chose FLOSS in the first place. However, while a rejection of your enthusiastic contributions can seem dictatorial and rude, it can also demonstrate a long-term, cohesive vision—a FLOSS community at its best.

This research was partially supported by NSF Grants 03–41475, 04–14468, and 05–27457. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Kevin Crowston *is an associate professor at Syracuse University's School of Information Studies. Contact him at crowston@syr.edu.*

James Howison *is a doctoral student at Syracuse University's School of Information Studies. Contact him at james@howison.name.*

Editor: Richard G. Mathieu, Dept. of Computer Information Systems and Management Science, College of Business, James Madison Univ., Harrisonburg, VA; mathierg@jmu.edu