Sustainability in Open Source Software Commons: Lessons Learned from an Empirical Study of SourceForge Projects

Charles M. Schweik

“*The real free-rider problems in open-source software are more a function of friction costs in submitting patches than anything else. A potential contributor with little stake in the cultural reputation game... may, in the absence of money compensation, think "It's not worth submitting this fix because I'll have to clean up the patch, write a ChangeLog entry, and sign the FSF assignment papers...". It's for this reason that the number of contributors (and, at second order, the success of) projects is strongly and inversely correlated with the number of hoops each project makes a contributing user go through.*”

Eric Raymond
Computer programmer, author, and open source advocate
in *The Cathedral and the Bazaar*

In this article, we summarize a five-year US National Science Foundation funded study designed to investigate the factors that lead some open source projects to ongoing collaborative success while many others become abandoned. Our primary interest was to conduct a study that was closely representative of the population of open source software projects in the world, rather than focus on the more-often studied, high-profile successful cases. After building a large database of projects (n=174,333) and implementing a major survey of open source developers (n=1403), we were able to conduct statistical analyses to investigate over forty theoretically-based testable hypotheses. Our data firmly support what we call the conventional theory of open source software, showing that projects start small, and, in successful cases, grow slightly larger in terms of team size. We describe the “virtuous circle” supporting conventional wisdom of open source collaboration that comes out of this analysis, and we discuss two other interesting findings related to developer motivations and how team members find each other. Each of these findings is related to the sustainability of these projects.

Introduction

This special issue of the *TIM Review* is devoted to questions surrounding the idea of “sustainability” in relation to open source software. The call for papers asked authors to connect some of Elinor Ostrom's work (1990: tinyurl.com/b3neybk; 2005: tinyurl.com/aesc7vd; 2010: tinyurl.com/aasko9e) related to sustainability, collective action, and the commons and apply it to open source. Over the last seven years, my research team and I have been doing just that. In this article, I summarize how we connected to Ostrom's approach to studying the commons and report some of the important findings related to questions of sustainability in open source software commons. The article focuses on the practical implications of the research findings.

Sustainability in Open Source Software Commons

Charles M. Schweik

Our Research Perspective

The overarching research question driving our research is: *What factors lead some open source software commons to success and others to abandonment?*

At the heart of this question is sustainability of open source software, from a *collaboration* perspective. Why do some programmers stay with a project while others leave? Here we focus not only on open source volunteer programmers – a central theme in many previous studies of open source – but paid programmers as well. Further, a central goal of our work was to investigate not simply high-profile, large-scale success stories (e.g., Linux, Apache Web Server), as was the case with much of the early research on open source, but to get a better handle on the unknown population of open source software projects, which at the time we started our work (~2005) was certainly well over 100,000 in number.

To begin our research, we built upon Elinor Ostrom and colleague's Institutional Analysis and Development (IAD) framework (Ostrom, 2005: tinyurl.com/aesc7vd; Figure 1). In this framework, as it applies to open source software commons, a central unit of analysis is the individual open source developer (diamond in Figure 1) who we assume is a boundedly rational actor and who periodically reflects on whether or not they should continue contributing to the project. This logic, at any point in time, is based in part on three groups of variables or influential factors that might contribute influence the developer's decision, depicted on the left hand side of Figure 1: i) Technological, ii) Community, and iii) Institutional attributes of the open source software project. In Schweik and English (2012; tinyurl.com/ap6cxuw), we review a significant amount of theoretical and empirical literature in an effort to identify important factors that are thought to influence other types of commons (such as natural resource commons) or are

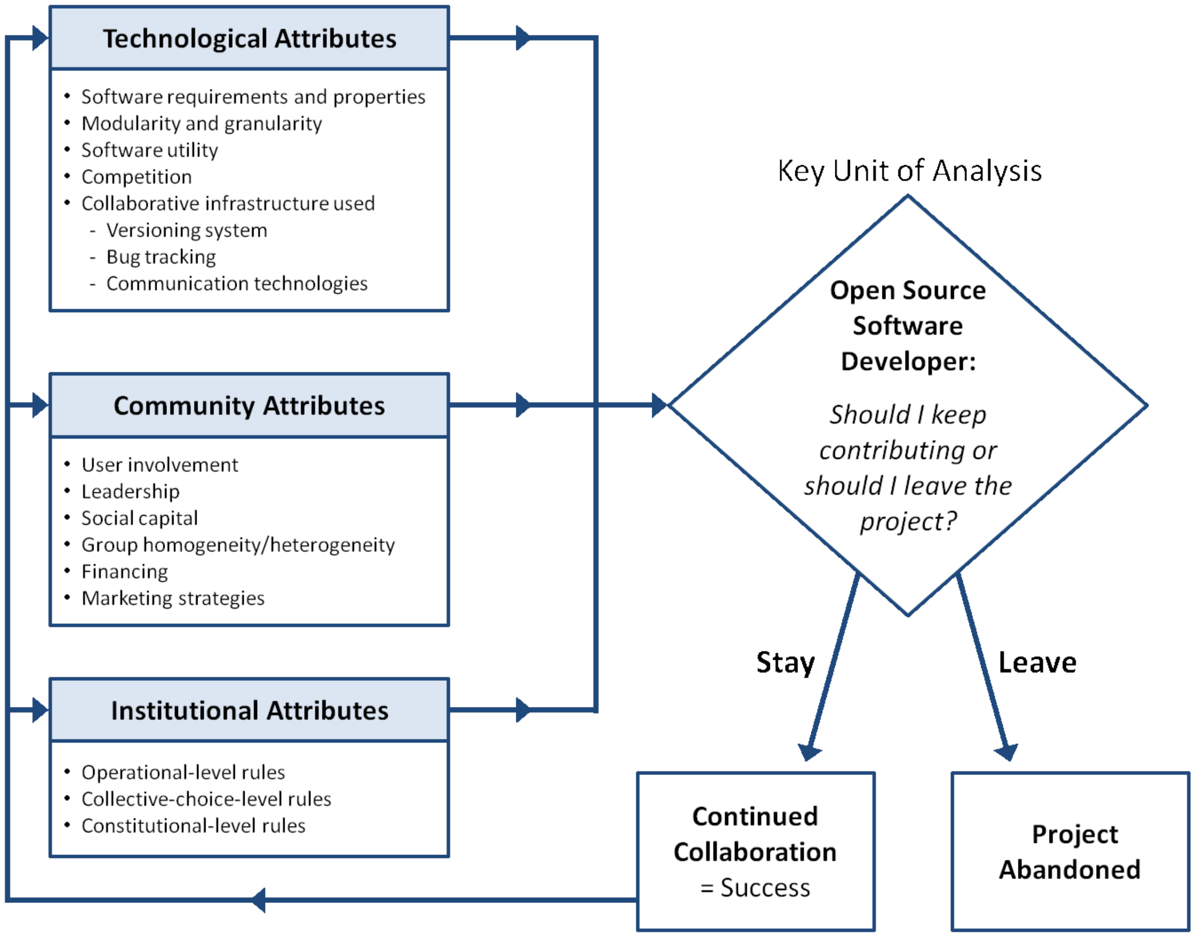
Technological Attributes
• Software requirements and properties
• Modularity and granularity
• Software utility
• Competition
• Collaborative infrastructure used
- Versioning system
- Bug tracking
- Communication technologies
Community Attributes
• User involvement
• Leadership
• Social capital
• Group homogeneity/heterogeneity
• Financing
• Marketing strategies
Institutional Attributes
• Operational-level rules
• Collective-choice-level rules
• Constitutional-level rules
Key Unit of Analysis
Open Source Software Developer:
Should I keep contributing or should I leave the project?
Stay
Leave
Continued Collaboration = Success
Project Abandoned

Figure 1. A simplified institutional analysis and development framework to support analysis of sustainability in open source software commons. Adapted from Ostrom (2005; tinyurl.com/aesc7vd) and Schweik & English (2012; tinyurl.com/ap6cxuw).

thought to influence the sustainability of software projects. This included literature specifically on open source, but also software engineering, virtual teamwork, and environmental commons or common property (e.g., forests, fisheries, irrigation systems). The three groups of attribute on the left side of Figure 1 list some of the factors – but not all – we identified through this work. To give the reader an idea of these three attribute groupings, let us consider an example of each.

A Technological Attribute thought to influence a developer's decision to stay with a project or leave might be related to "task granularity" as Yochai Benkler (2006; tinyurl.com/6ftot3) puts it; if the development task is too large or "coarse grained", the developer might decide it requires too much effort for the volunteer (or paid) time he or she can allocate to it and might decide to leave the project.

A Community Attribute thought to influence a developer's decision to stay or leave might be the attributes of the leader(s) of the project. Leadership is a complicated variable or set of variables, but one aspect of it relates to the idea of leading by example; leaders motivate others on the team to do work by contributing significant work themselves.

An Institutional Attribute thought to influence a developer's willingness to stay with a project or leave might be the level of formality required to participate on the project. A famous proponent of open source, Eric Raymond (2001; tinyurl.com/d546xlv) described formalized rules for collective action in open source as "friction" that creates negative incentives for contribution (see the introductory quote above). Space limits us to describe all the variables we investigated in this study, but the topics listed in the three boxes on the left side of Figure 1 will give the reader a sense of the kinds of variables we investigated. Ultimately, we identified over 40 variables, most of which led to testable hypotheses where *a priori* expectations on their influence were known. However, in some cases, we had no idea what relationship would be found, and no previous theory or empirical work to suggest an expected relationship with our dependent variable, success or abandonment of open source software projects.

The reader should note that Figure 1 represents a dynamic system that changes over time. As long as a project stays operational, there is feedback threading back to the three sets of attributes to the left in Figure 1, and periodically, these attributes might change in some dimension. These changes then have an effect or may influence the developer's feelings about the project and their periodic reflections on whether to stay or leave, and the cycle continues.

Methods

To begin our empirical work, we first searched for a dataset on open source software projects that was already collected, rather than having to build one from scratch. Fortunately, a group called FLOSSMole (flossmole.org) based out of Syracuse University had been actively scraping the dominant open source project hosting site SourceForge (sourceforge.net) and building a database on these projects for other researchers to use (Howison et al., 2006; tinyurl.com/abounnq). Their database contained metadata about these projects, most related to Technological or Community-related attributes, but with at least one Institutional variable (license used). Our initial SourceForge database, gathered in the summer of 2006, contained 107,747 projects. In 2009, we collected a second time-slice from a different repository called the SourceForge.net Research Data Archive (tinyurl.com/ard7v9z), which is housed at the University of Notre Dame. This second dataset, representing SourceForge projects in 2009, contained 174,333 projects.

Our next step was to formulate a measure of success and abandonment for open source software projects. This was a challenging endeavour, which took us over a year and a half to complete. We first identified two different longitudinal stages that open source projects go through: i) an Initiation Stage and ii) a Growth Stage. The Initiation Stage describes the period of time from project start to the first public release of software. On the SourceForge hosting site, it is easy to find new projects that have yet to make code available to the public but are being actively worked on. We use the Growth Stage to describe the period after a project's first public release of code. One could conceptualize a "termination" or "abandonment stage" as well, but in our conceptualization, that particular event can occur in either the Initiation Stage (pre-first release) or in the Growth Stage (post-first release).

With these two stages defined, we then set out to carefully define, both theoretically and empirically, a method to measure whether a project is successful or abandoned in these two stages. We identified six categories of success and abandonment: Success in Initiation (SI); Abandonment in Initiation (AI); Success in

Growth (SG); Abandonment in Growth (AG); Indeterminate in Initiation (II); and Indeterminate in Growth (IG). Details of this initial phase of our research can be found in English and Schweik (2007; tinyurl.com/bd29rnu). Our classification system was later replicated independently by Wiggins and Crowston (2010; tinyurl.com/a33k9fn). Table 1 presents our definitions and results for the 2006 SourceForge dataset; for the results from our 2009 SourceForge data, please see Schweik and English (2012; tinyurl.com/ap6cxuw).

These datasets provided an excellent start, but our mapping of SourceForge projects to the identified theoretical variables (Figure 1) led to the conclusion that many of the community and institutional variables we wanted to investigate were not captured in these datasets. Consequently, in 2009, we implemented a complementary online survey for SourceForge developers to capture these missing variables. The challenge was that, if we contacted a random sample of SourceForge project administrators, we expected that we would get significant bias toward successful collaborations that were active. To ensure enough responses from abandoned projects, we needed to sample a much larger number of SourceForge projects. In the summer of 2009, we stratified our 2009 dataset using our success/abandonment classification and randomly selected 50,000 projects to survey. With the help of the SourceForge organization, we emailed a survey to the SourceForge project administrators for each of these projects. The result: 1403 surveys returned.

Table 1. Success and abandonment categories for open source software projects in the 2006 SourceForge database

Class	**Definition**	**Number of Projects***
SI: Success in Initiation	Developers have produced a first release.	–
AI: Abandonment in Initiation	No first release produced, and the project appears to be abandoned.	37,320 (35%)
SG: Success in Growth	Project has achieved 3 meaningful releases of the software and the software is deemed useful for at least a few users.	15,782 (15%)
AG: Abandoned in Growth	Project appears to be abandoned before producing 3 releases of a useful product, or has produced 3 or more releases in less than 6 months and is abandoned.	30,592 (28%)
II: Indeterminate in Initiation	Project has yet to reveal a first public release but shows significant developer activity.	13,342 (12%)
IG: Indeterminate in Growth	Project has not yet produced three releases but shows development activity, or has produced three releases or more in less than six months and shows development activity.	10,711 (10%)
Total projects		**107,747**

* Successful Initiation (SI) numbers are not listed because these successes are Growth-Stage projects; including the SI category would double-count projects.

With the online survey conducted, we were able to create a database of these 1403 projects and combine it with the SourceForge metadata from the 2009 Notre Dame dataset. We had a complete dataset capturing both our dependent variable of success and abandonment for all Initiation Stage and Growth Stage projects, as well as measures for our independent variables related to Technological, Community, or Institutional attributes. The dataset captured more than 40 independent variables, a small sample of which are listed in Figure 1.

We used three statistical techniques to analyze the data. To investigate relationships of individual variables, we used contingency tables to investigate the differences in distribution for the projects as they relate to success and abandonment. We also used two different multivariate analytic techniques: i) classification and regression trees and ii) logistic regression. Full explanations of these techniques, as well as summary tables and results are available in Schweik and English (2012; tinyurl.com/ap6cxuw).

Selected Findings

Our analysis focused on over 40 variables thought to potentially influence whether open source projects maintained collaboration or whether they became abandoned. In this section, we will focus on some of our more general or most interesting findings, with a focus on practical insights.

First, we have empirical support for the conventional thinking of how open source software projects operate. The vast majority of open source projects do not have large teams, but rather have very small teams of one to three developers. Based on careful analysis of both Initiation Stage and Growth Stage data, we found that the majority of these projects tend to start with a very small development team of one to two developers and very little or no user community. Then, as work progresses and after a first release is made, a user community is established and grows over time. The founding developer(s) lead through doing, and through the development of a product that they often need (supporting von Hippel's [2005; tinyurl.com/57xp5x] idea of “user-driven need”), build something usable and, at the same time, begin to generate a user community. Through the regular open source communication channels (e.g., IRB sessions, email lists, websites, and bug tracking systems), they build social capital between themselves and their user base, and gradually grow their user base, and a virtuous cycle begins. More progress is made on the code base, leading to (potentially) a larger user base, and leading to (perhaps) an added developer. But, our study may be some of the first empirical results that actually capture this conventional thinking of how open source collaboration operates.

We also discovered that the successful Growth Stage projects tend to gain one developer compared to abandoned ones and, to our surprise, we found that over 58% of our successful projects gained a developer from *another continent*. This last point is quite striking, for we found that in many cases these new developers have never met face-to-face in person with other developers on the project but know and trust each other as a result of almost strictly Internet-based interaction. These findings align with what we have heard from open source developers we have interviewed.

Based on what we have found, related to the idea of open source project sustainability, the advice we have for leaders of projects in the Initiation Stage is:

1. Be ready to put in the hours. Work hard toward the creation of the first software release.

2. Demonstrate and signal good leadership by administering your project well and clearly articulating your vision and goals through project communication channels (e.g., website, bug tracking system). Create and maintain good documentation for potential new developers and for your user community through these channels.

3. Advertise and market your project and communicate the plans and goals, especially if you seek new developers to move the project forward over the longer term.

4. Realize that, in our data, successful projects are found in either GPL-compatible or non-GPL-compatible free/libre open source licenses.

5. When starting a project, consider its potential to be useful to a substantial number of users. The more potential users you have, the higher the likelihood that one or more of those users will have relevant skills and interests to consider joining and contributing to your project down the road.

Sustainability in Open Source Software Commons

Charles M. Schweik

Our advice for leaders of projects in the Growth Stage (post-first release) includes:

1. Focus on the idea of creating and maintaining the "virtuous circle", where good initial products attract users, which then potentially attract a new developer, which leads to more improvements. Our research clearly shows that successful projects have a potentially significant user community and that this user community drives project continuity.

2. Make sure that there are tasks of various sizes or effort demands that people can contribute to. Successful Growth Stage projects tend to have tasks for people to work on that fit into their available schedules. We remind readers of the concept of task granularity by Benkler (2006; tinyurl.com/6ftot3), mentioned earlier.

3. Surprisingly, our data suggests that competition seems to favour success rather than hinder it. In other words, do not give up if some competition appears on the horizon.

4. Financing helps.

5. To the extent possible, keep rules governing project collaboration and project governance lean and informal. To a large measure, the operational rules that do exist in open source software projects are often embedded in the version control systems that support the projects (e.g., CVS, Subversion), or are simple group-established social norms. We found that the vast majority of the projects we studied had very little *formalized* governance and operated under "Benevolent Dictator" type governance structures. In other words, they tend to support our opening quote by Eric Raymond (2001; tinyurl.com/d546xlv). Our sense from our study that simple, agreed-upon norms tend to drive these projects is in part because the vast majority of the projects we studied are very small teams that need very little in terms of formal coordination. However, we did have evidence that, as teams increase in size, project governance moves toward more formalized systems. Our evidence is fairly limited because, in our dataset, a very small proportion of the projects studied had large teams with 10 or more developers. But, this suggests that, if a project team grows, the team should not hesitate to move toward more formalized systems if required.

Our data analysis also led to some theoretical findings related to sustainability of open source software projects. The two most interesting of these findings are described below.

1. Developer motivations

Regarding questions of why developers participate in open source software projects, our results support much of the existing empirical work done earlier. Across both abandoned and successful projects, a primary motivator for participation was von Hippel's (2005; tinyurl.com/57xp5x) user-centric need. Developers participate because they themselves are users of the software or because the organization they work for depends on it. Other developers participate because they learn from the process of reading others' code and then developing new functions for the product. Others participate as a kind of "serious leisure" where they use their programming skills that they use for their employment and apply it to something outside of their work domain for their enjoyment. The one motivation that past research has suggested is important – that we found was not important – is the idea of signaling programming skills to others, often in an effort to possibly find eventual employment. In our survey data, this was not reported as an important factor and, in our view, it is because the vast majority of the teams are quite small (i.e., 1–3 people). But, perhaps the most interesting and new finding regarding motivations for participation in our research is our finding that projects with developers who have *multiple motivations* driving their participation will be more successful than projects with developers with only one motivation. In other words, open source projects will be more sustainable if individual members on the team have multiple reasons (e.g., "I learn and am paid to participate", or "I contribute because I am contributing to a public good and because I enjoy working on the project") driving their interests to contribute.

2. Sourceforge and Google as intellectual matchmakers

Some of our most careful work in this study revealed that successful open source software projects gain a developer and that quite often this new developer is not physically co-located with the developer(s) who founded the project, but rather, are geographically distant, and often on another continent. This provides some strong evidence suggesting that well-known websites for open source software, such as SourceForge, coupled with web search engines such as Google, create an intellectual matchmaker of sorts through “power-law typology” (Karpf, 2010; tinyurl.com/b6cxpzb). These power-law hubs are locations on the Internet that provide value to their users in part because of the network effects created because they have large crowds of similar users. Regardless of where a programmer lives in the world, people can find software projects that are related to this need and, over time, build social capital with the developers and eventually join the team if they speak the same language and demonstrate the desire and the skills needed to collaborate.

Conclusion

In this article, we described a five-year US National Science Foundation research study on the factors that lead some open source projects to ongoing collaboration and others to abandonment. To summarize, we find strong empirical support for the conventional wisdom of how open source software projects are sustained (see the virtuous circle discussion above) and report two of the most interesting findings of the study: i) that projects will be more sustainable if developers have multiple incentives driving their participation; and ii) successful projects gain a developer and this is likely driven through the intellectual match-making created by search engines such as Google coupled with power-law hubs such as SourceForge. For more detail on the research reported here, see Schweik and English (2012; tinyurl.com/ap6cxuw).

Acknowledgements

This research was supported by the U.S. National Science Foundation under grant number 0447623. The findings, recommendations, and opinions expressed here, however, are my own and do not necessarily reflect the views of the NSF. I am also very grateful to my research team: Robert English, Sandy Haire, Meng-Shiou Shieh, and others, without whom this work would not have been accomplished.

About the Author

Charles M. Schweik (Charlie) is an Associate Professor with a joint appointment between the Department of Environmental Conservation (eco.umass.edu) and the Center for Public Policy and Administration (masspolicy.org) at the University of Massachusetts Amherst. He is Associate Director of the National Center for Digital Government (ncdg.org) and the founding member of a new “Workshop on the Study of Knowledge Commons” on campus. His research focuses on environmental management and policy, public-sector information technology, and the intersection of those domains.

Citation: Schweik, C. M. 2013. Sustainability in Open Source Software Commons: Lessons Learned from an Empirical Study of SourceForge Projects. *Technology Innovation Management Review.* January 2013: 13-19.