Open Source: From Mythos to Meaning

Alexander C. MacLean and Charles D. Knutson

Computer Science Department, Brigham Young University, Provo, Utah
amaclean@byu.edu, knutson@cs.byu.edu

Abstract. Free open source software (FOSS) projects expose rich development, evolutionary, and collaborative data from which researchers have formed theories and conclusions about the FOSS development ecosystem. However, little work has been done to determine whether FOSS projects are analogous to proprietary development efforts. We propose several axes along which taxonomies of FOSS and proprietary projects may be created and compared, and preview several future studies that will begin to populate these taxonomies.

1 What is “Open Source”?

Open Source proponents have long touted the advantages, be they financial, social, ethical, or moral, of developing software using an “open source” paradigm. However, even among some of the earliest players in the “open source” space, there is little agreement about its exact definition. Some claim that open source is a convenient (and possibly more efficient) way for developers to collaborate on major software projects.

Linus Torvalds: Me, I just don’t care about proprietary software. It’s not “evil” or “immoral,” it just doesn’t matter. I think that Open Source can do better... it’s just a superior way of working together and generating code.

It’s superior because it’s a lot more fun and because it makes cooperation much easier (no silly NDA’s or artificial barriers to innovation like in a proprietary setting)... [24]

Others elevate the paradigm to pseudo-religious stature.

Richard Stallman: We like to think that our society encourages helping your neighbor; but each time we reward someone for obstructionism [not sharing code], or admire them for the wealth they have gained in this way, we are sending the opposite message [21].

Still others herald the potential business advantages.

Eric Raymond: [Open Source is] the process of systematically harnessing open development and decentralized peer review to lower costs and improve software quality [19].
Bob Young: *Open Source* gives customers control over the technologies they use, instead of enabling the vendors to control their customers through restricting access to the code behind the technologies [19].

Others are confused by what we’re even talking about in the first place.

Inigo Montoya: *You keep using that word. I do not think it means what you think it means.*

It is clear that open source has changed the landscape of software development. But what do we, as researchers, mean when we refer to “open source”? You may notice that the preceding quotations aren’t actually definitions, but rather descriptions of the attributes or benefits of an “open source” product or organization. The problem is that when we conduct research, loose definitions and vague conceptual notions aren’t good enough.

To illustrate the point, here are several definitions of “open source” we have heard and/or used while discussing the open source movement with other researchers:

1. A licensing model that requires redistribution of code along with a product.
2. A mythos concerning the operation and constitution of open source communities that encourage volunteer participation by developers.
3. A mythology concerning the origins of open source that contrasts open source operations with traditional, closed source operations.
4. A convenient licensing model that allows organizations to collaborate on infrastructure (plumbing, if you will) while differentiating themselves in higher level software.
5. A direct attack on “the man” and the closed source restrictiveness of imperialist software companies.
6. A marketing ploy by large corporations to engender good will with customers.
7. Any number of other definitions, depending upon the circumstance and audience.

In each case (with the possible exception of Item 5), the definition is sufficient for the particular case to which it is applied. However, none of these is broad enough to capture the many variations of the open source paradigm.

A taxonomy that considers open source development organizations, as well as the developers and organizations that contribute to open source projects, would allow researchers to qualify results within the confines of the taxonomy of the organization from which they were drawn, rather than from an arbitrarily broad set of ambiguous definitions. Moreover, a taxonomy would ground research findings within a common theoretical framework and provide a mechanism for determining the degree to which such results can be extrapolated to other projects and organizations (whether open or closed source).
2 Open Questions

For the moment, let us ignore the vague definitions and distinctions of open and closed source. Instead, let’s start with a broad definition of open source software that simply requires that the source code is available to the end user... eventually\(^1\), and refer to this as Free and Open Source Software (FOSS). This definition only distinguishes between organizations that restrict access to their source code and those that don’t.

Definitions:

**FOSS:** Software for which the source code is eventually available to the user.

**Proprietary Software:** Any software that is not FOSS.

Although some work has been done to explore the behaviors of open and closed sourced development organizations, little effort has been expended to understand the differences between the developers in such organizations. Many questions persist:

1. Who are the developers who spend their time working on these projects?
2. Which organizations employ open source developers, and what are their motivations? Although studies have examined this question, more work is required to build a taxonomy of the results.
3. Are open source developers somehow different from those that work on closed projects? More formally, what is the taxonomy of developers who choose to (or are employed to) work on open source projects?
4. Is the taxonomy of open source developers significantly different than the taxonomy of the general population of software developers? If so, how? Along what axes are these two taxonomies analogous?
5. What does it mean if open source developers are not significantly different from the general population of developers (along certain axes)?
6. What does it mean if open source developers are significantly different from the general population developers (along certain axes)?

In the following subsections, we address some questions and posit theories that arise from these potential lines of inquiries. In Section 3 we propose methods for answering some of these questions.

2.1 The FOSS Developer

Who are these developers that work on FOSS projects? Bird, et al. report that much of Eclipse is written in-house at IBM. On the other hand, “\(^{1}\) The “...eventually” clause in this definition is required because the publicly available trunk for Android, a major open source project, is typically several months out of date.
to Firefox come from a myriad of sources and no single commercial entity completely controls or owns the development process” [1]. A more complete understanding of the developer taxonomy would allow researchers to design better studies and draw more accurate conclusions about the state of FOSS and proprietary development organizations and practices. In this section we list several axes of a potential taxonomy of FOSS developers.

**Developer Motivation** Lakhani and Wolf, in a study of 684 software developers in 287 FOSS projects, found that the key factor in developer participation was “enjoyment-based intrinsic motivation.” However, they did not take into account the power law distribution of developer contributions which describes a common phenomenon in FOSS projects: a small (relative to the project size) set of core developers often develop most of the functionality (see Figure 1). Many questions remain regarding the core developers within FOSS organizations. For example, are the central figures within these projects paid, while the ancillaries are motivated by a desire for creative outlet?

![Fig. 1. Developer commit volume on the Apache HTTP Server project. The x-axis is individual developers, sorted by commit volume.](image)

**Job Tenure** During the dot com boom, conventional wisdom held that a developer remained in a job for 18 months. By 2003, three years after the bubble burst, IT workers who earned Computer Science degrees had an average job tenure of 6.2 years [5]. In contrast, the median tenure of developers on the Apache HTTP Server project is 3.7 years², with a strong right tail (see

---

² Tenure in this case is defined as the length of time between a developer’s first and last commit. Potential issues with this definitions are addressed in Section 3.
Figure 2), just more than half the tenure of the general developer population. However, the top 15 committers to the Apache HTTP Server project have a median tenure of just under 8 years.

![Graph showing tenure distribution](image)

**Fig. 2.** Tenure of developers on the Apache HTTP Server project.

Job tenure is illustrative of expertise and seniority in a development project. Tenure could vary based upon differences between an open source meritocracy and traditional, corporate approaches.

**Developer Background (Work Related)** Two developer characteristics related to job tenure—age distribution and industrial tenure—reveal some measure of industrial, commercial, and organizational expertise. Other attributes, such as education, illustrate the ability of an organization to attract top talent. With respect to age distribution and industrial tenure, a major question arises: is open source development dominated by youthful exuberance or aged wisdom? Or is it composed of a healthy amalgam? We propose three theories based upon the potential answers to the preceding questions.

Ghosh, et al. found that in 2002 the median age of FOSS developers was 26 and that only 10% were older than 35 [7]). If youth typifies the development environment, we theorize that it reflects two attributes of the open source economy:

1. Organizations that support FOSS projects view the projects as plumbing, not an area in which to differentiate themselves, and therefore allocate less-experienced developers, and/or
2. Outside of regular employment, younger developers are more likely to have the free time required to make meaningful contributions to FOSS projects.
Lakhani and Wolf found that “formal IT training... reduces the number of hours spent on a project” [11]. In their study, this result was incidental, and therefore received little attention. Nevertheless, it leads to questions related to the role of industrial experience in FOSS contribution and the shape of the FOSS developer taxonomy. However, the aforementioned age distribution does not consider developer prominence or role in an organization. On the other hand, if FOSS projects, or more importantly, leadership within FOSS projects, are the purview of elder hackers, we theorize contrasting economic attributes:

1. Organizations depend upon the success of a FOSS project in order to differentiate their services or products, and therefore assign high priority to its development, and/or
2. Older, more experienced, developers find time to contribute, even when not compensated.

Of course, neither of these two polarized theories is likely to be found entirely applicable in all projects. Instead, a mixture may exist where, for example, organizations tend to employ older developers, while younger developers tend to have the flexibility to contribute code on their own time. Whatever the distribution across developer age and experience, a more clear understanding would provide insight into the motivation and monetization structures of companies that contribute to open source projects, as well as to the experience level of the developers.

If the distribution of developer age and experience does not, in fact, gravitate towards one of these poles, it could indicate that leaders of these organizations don’t discriminate for or against their FOSS collaborations, but instead view them as parallel and complementary to their other development efforts.

**Developer Background (Non-Work Related)** Gender, race, and other personal attributes are discoverable in a typical development organization. In fact, even in proprietary, distributed development organizations, these attributes are known because hiring and promotion interviews are still performed face-to-face. On the other hand, unless explicitly exposed by the developer, these attributes are largely latent in meritocratic organizations such as the Apache Foundation. Other attributes, such as socio-economic background, familial status (single, married, married with children, single parent, etc.), and religion, are often latent in both types of organizations.

A taxonomy and understanding of the personal attributes of developers within a meritocratic FOSS development organization and within proprietary organizations would allow us to analyze whether the latent nature of these attributes protects FOSS projects from flexible definitions of merit [25] and other types of discrimination. A positive result could suggest a fascinating feature of distributed, FOSS development: insulation from discriminatory practices, both intended and unintended [26].
Specialization and Private Information

Previous work indicates both the presence of private information\(^3\) in large organizations [10] and a lack of specialization (a subtype of private information) in the Apache HTTP Server project [13]. Lack of private information and specialization in open source development organizations contrasts starkly with many proprietary environments where specialization is often associated with efficiency and job security. If this contrast holds across open development organizations as a whole, it represents a fundamental difference in both communication requirements and organizational behavior.

Programming Languages

Although not a core developer attribute, differences in programming language fragmentation, or the degree to which a developer utilizes more than one programming language, would illustrate differences between imposed technical environments and less-structured, self-organizing, distributed communities [8, 9]. We suspect that along this axis, FOSS and proprietary organizations will appear similar (organizations must standardize upon some set of tools and languages in order to be productive, whether FOSS or proprietary). However, FOSS developers who work on multiple projects may have higher language fragmentation than proprietary developers who only work on a single project.

Salary

Ghosh, et al. note that 52% of FOSS developers (in 2002) earned no more than 2,000 U.S. Dollars or Euros per month [7]. In contrast, Choy et al. reported that the average monthly income for all U.S. Computer Science graduates 10 years after graduation\(^4\) (in 2003) was 6,050 U.S. dollars [5]. This disparity is exaggerated by a high level of student contributions to FOSS projects: 17%. Also, only 14% of respondents live in North America, so it would be unwise to make direct comparisons between the two studies. However, the large difference certainly suggests inequalities in our open and closed source taxonomies.

Or does it? Ghosh, et al. don’t take developer prominence or position into account. Although many of the developers on the Apache HTTP Server project are most likely students, it does not follow that the most or any of the 15 core developers are as well [16]. Salaries for core FOSS developers may be on par with their proprietary counterparts. If so, this seeming discrepancy fades in the light of taxonomic clarity.

2.2 Contributing Organizations

Many organizations that build proprietary products also contribute to FOSS projects [4].

\(^3\) We utilize a definition of private information as “...the challenge of utilizing distributed knowledge in an organization...where private refers to information possessed by a relatively small segment of the population—as opposed to information that is widely held” [10].

\(^4\) According to Ghosh, et al., the median age of FOSS developers is 26 [7], close to the average age of computer science graduates 10 years after graduation.
Motivations  Lerner and Tirole present three motivations for organizations to expend resources on products that don’t directly generate revenue: 1) Living symbiotically off an open source project (Red Hat, Sun, and Oracle); 2) Code re-release to benefit a complementary market segment (Hewlett-Packard and IBM); and 3) Acting as intermediaries (CollabNet) [12]. In essence, the existence of the product produces side effects that positively affect the bottom line of the company.

Capek, et al., in explaining the genesis and integration of open source ideals at IBM, stated “....open source did not pose an immediate threat to our existing business, and in fact, our products could benefit from supporting and building open source.” They note that participating in open source projects yielded two key benefits (for IBM) [4]:

1. Reusing open source components decreased overhead versus building proprietary solutions. Example: using the Apache HTTP Server as a key component of the WebSphere Application Server.
2. Collaborating with others in the community to develop necessary but low margin tools frees up resources for more lucrative projects. Example: Eclipse.

In 1996, Apple acquired NeXT in an effort to modernize its aging operating system and salvage its dwindling market share. The company then began using NeXT’s BSD-based Unix variant, NextStep, as the base for its operating system. It released the core components of its operating system as a BSD-licensed FOSS project named Darwin, but kept the GUI and many of the APIs (including the Java API) proprietary. This “layered” open-closed approach allowed Apple to reap the benefits of using a proven FOSS technology while maintaining control over many of the distinguishing components of its operating system [28].

Additionally, Sun slowly moved into FOSS in an attempt to stymie advances by Microsoft and Linux [28].

Organizational Attributes  As with developers, understanding the taxonomical attributes of organizations that contribute to FOSS ventures—such as size, revenue, location, and industry—would allow researchers to draw parallels to the general population of organizations. In addition, such a taxonomy would provide a standard against which to measure claims that attempt to extrapolate results from open source projects.

2.3 The FOSS Organization

A taxonomy of developers is incomplete unless married to a taxonomy of the structures within which they operate. On the surface, FOSS and proprietary development patterns may appear different (methods of joining the organization and metrics for defining prominence, for example). However, Mockus, et al. describe developer communication and collaboration patterns in the Apache HTTP Server and Mozilla Firefox that sound very similar to those found in
modern proprietary development organizations of comparable size [17]. In interviews with FOSS developers, Schweik and English found similar results [20].

Commit Patterns Organizational commit patterns are indicative of types and levels of developer collaboration. Monolithic commits may cause sweeping changes, while small commits indicate incremental development. In Production/Stable or Maintenance phase projects on SourceForge we found both patterns of small commits and patterns of large, monolithic commits [14, 15]. Further work has refined our notion of the reasons behind large commits through the development of a taxonomy of large commits [18]. However, correlation between commit behaviors in FOSS projects and commit behaviors in proprietary projects has not been adequately defined.

File Level Collaboration and Code Ownership Bird, et al. demonstrated that, in Windows Vista, Mozilla Firefox, and Eclipse 1) “software components with many minor contributors will have more failures than [those] that have fewer” and 2) ownership only had a consistent, statistically significant effect on Windows Vista [1]. In addition, previous studies have uncovered patterns of author contributions that may or may not be analogous to those in proprietary development [22, 23].

Communication Patterns Crowston and Howison found that FOSS projects on SourceForge exhibit myriad communication patterns. They contend that employing metaphors such as “the cathedral and the bazaar” [19] doesn’t capture the complexity and variance within the many projects on SourceForge [6].

3 Toward a Taxonomy

We propose to begin to develop a taxonomy and theoretical framework of FOSS developers and organizations. This framework should provide a common foundation upon which researchers may base their dialogue. Specifically, we will create sub-taxonomies of FOSS and proprietary projects along the following axes 1) Job Tenure, 2) Developer Specialization, and 3) Code Ownership. In addition, we propose future work developing taxonomies of Work Related Developer Background and Non-Work Related Developer Background.

3.1 Data Sources

We use several data sources to draw conclusions about the state of these development communities: 1) the Current Population Survey, 2) a local, small company with which we have a research relationship, 3) two potential large companies, and 4) online FOSS repositories.
Current Population Survey  Aside from the decennial census, the U.S. Census Bureau compiles a monthly Current Population Survey. The Bureau collects survey responses regarding numerous subjects, ranging from labor to living conditions, from a “multistage stratified sample of approximately 72,000 assigned housing units from 824 sample areas” [3]. Most importantly, this survey collects information relating to job tenure, job satisfaction, occupation, job industry, salary, number of hours worked at each job that a respondent holds, race, age, familial status, living conditions, and religion.

Small Companies We have affiliations with a small (around 30 developers) local company with whom we have run organizational studies. This company has been a leader in its market for over 20 years and employs both long tenured and newly hired developers.

Large Partners We have relationships with two large development organizations that have research divisions. In both cases, we should be able to leverage those relationships to develop taxonomies of large, proprietary organizations and their developers.

FOSS Repositories Publicly available data from the Apache Foundation, the Eclipse Foundation, the Mozilla Foundation, SourceForge, other FOSS “forges,” and the SourceForge Research Data Archive [27] provide rich information about developer interaction and productivity. Many of the studies in Section 2 used this data [1, 6, 8, 9, 14, 15, 16, 18, 22, 23].

3.2 Job Tenure

Determining job tenure from census data is straightforward, as is supplementing the census data with data from companies with which we have research affiliations. In contrast, before attempting to determine job tenure in a FOSS project, we must first define what “tenure” means. If a developer commits a patch, takes ten years off, and then commits another patch, is that considered a ten year tenure? Common sense says no. But where does one draw the line?

Instead of a single definition of job tenure, we propose a taxonomy of FOSS developer tenure that incorporates time spent on a project, productivity during that time period, periods of inactivity and developer centrality. We can draw this data from the commit logs and mailing lists of online FOSS communities.

3.3 Code Ownership

Previous studies have correlated code ownership with defects [1] and have explored patterns of code ownership in SourceForge [23], Eclipse projects [22], Mozilla Firefox, and Windows Vista [1]. The taxonomy of code ownership would contain attributes such as proportion of files owned by a single author, distribution of file ownership measured through authorship entropy [23], density of commits to a file, and frequency with which a developer replaces lines of code contributed by other, less prominent developers.
3.4 Developer Specialization

Developer specialization can be gleaned from commit histories by identifying the modules to which developers commit code. Commits can then be assigned to categories such as UI development, database connectivity, and operating system integration. When possible, developer interviews provide fine-grained insight into the perceived specialization within the organization. Our analysis of specialization hinges on our collaborative research efforts with outside companies and the availability of FOSS repositories.

In addition to direct approaches for FOSS projects, we can also utilize indirect methods such as topic analysis of bug comments and mailing lists to identify developers within a FOSS organization who hold specialized information [2, 13]. Using this information, in conjunction with commit behaviors, should uncover developers who have specialized information, but who use that information in the aid of others in a mentoring or leadership capacity.

3.5 Future Goals

Although not yet formalized, and likely outside the scope of the work of a single doctoral student, taxonomies of developer background, both work-related and not, are vital to understanding the differences (if any) that exist between the population in the FOSS community and the developer community as a whole.

**Developer Background (Work Related)** Work-related developer background would require interviews with developers in both FOSS and proprietary organizations, a daunting, but achievable task.

**Developer Background (Non-Work Related)** Ironically, the data to determine non-work related background (socio-economic status, race, familial status, etc.) is freely available for the general population of developers in the United States through the *Current Population Survey* [3]. Ghosh, et al. provided survey results on some of these attributes for the FOSS community, but, as noted in Section 2.1, the granularity of the data is insufficient to build a taxonomy of developers. Determining these attributes for just FOSS developers would require personal interviews as well as very personal questions. While we would love to conduct these interviews, finding cooperative subjects may prove challenging.

4 Conclusion

Proclaiming the supposed advantages or disadvantages of FOSS and proprietary development paradigms ignores the rich and diverse attributes of development organizations. Likewise, qualifying research as resulting from analysis of open or closed source organizations both overgeneralizes by extrapolating to projects that may not be similar and misses possible applicability to similar projects.
that have different paradigms. In qualifying research results, the question is not “Are open and closed source projects the same?” Instead, we must ask several questions, such as

1. What are the taxonomies of FOSS and proprietary project?
2. How do the projects from which we draw our data fit into these taxonomies?
3. Now that we understand how these projects are similar to or different from other projects, how does that affect the interpretation of our results?

We propose taxonomies of FOSS and proprietary developers and organizations so that we, as a research community, can ground our findings within a common theoretical framework. Doing so will enhance the dialogue among researchers within the OSS research community as well as between the research community and development organizations, both FOSS and proprietary. The end goal of this effort is to more precisely define our lexicon and increase the practical applicability and acceptance of our work.

References


