2013-04-23

Prevalence of Reflexivity and Its Impact on Success in Open Source Software Development: An Empirical Study

Brandon D. Foushee
Brigham Young University - Provo

Prevalence of Reflexivity and Its Impact on Success in Open Source Software Development: An Empirical Study

Brandon D. Foushee

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Charles Knutson, Chair
Daniel Zappala
Eric Mercer

Department of Computer Science

Brigham Young University

April 2013

ABSTRACT

Prevalence of Reflexivity and Its Impact on Success in Open Source
Software Development: An Empirical Study

Brandon D. Foushee
Department of Computer Science, BYU
Master of Science

Conventional wisdom, inspired in part by Eric Raymond, suggests that open source developers primarily develop software for developers like themselves. In our studies we distinguish between *reflexive* software (software written primarily for other developers) and *irreflexive* software (software written primarily for passive users). In the first study, we present four criteria which we then use to assess project reflexivity in SourceForge. These criteria are based on three specific indicators: intended audience, relevant topics, and supported operating systems. Based on our criteria, we find that 68% of SourceForge projects are reflexive (in the sense described by Raymond). In the second study, we randomly sample and statically estimate reflexivity within SourceForge. Our results support Raymond's assertions that 1) OSS projects tend to be reflexive and 2) reflexive OSS projects tend to be more successful than irreflexive projects. We also find a decrease in reflexivity from a high in 2001 to a low in 2011.

Keywords: Eric Raymond, open source software, reflexivity, SourceForge, success factors.

ACKNOWLEDGMENTS

> *"Trim your feeble lamp, my brother;*
> *Some poor sailor, tempest-tossed,*
> *Trying now to make the harbor,*
> *in the darkness may be lost."*
> (Philip Paul Bliss, "Brightly Beams Our Father's Mercy" (No. 335), in Hymns of The Church of Jesus Christ of Latter-day Saints, 1985)

I thank all those whose light has helped me to finish this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Open Source

Open source is often referred to as a "movement" [24] [28], embodying a new way to develop software. It has been heralded as "a superior way of working together and generating code" [41]. Yet, for all the attention devoted by researchers to the "superior way" in which developers create OSS, considerably less attention has been paid to the *users* of OSS.

Open source evangelists, such as Eric Raymond, have repeatedly stressed the critical role of users in the process of OSS development. As an example, Raymond's articulation of Linus's Law, that "given enough eyeballs, all bugs are shallow," affirms the essential role that users play in OSS development. However, in the *Cathedral and the Bazaar*, Raymond consistently conflates users with developers, presumably because in Raymond's experience all users of his low-level software components were, in fact, developers [35]. Such fundamental confusion perpetuated by one of OSS's most foundational philosophers has contributed to a general failure by OSS researchers to distinguish between OSS developers and OSS users.

As use of OSS becomes more prolific, researchers are finding that most users are, in fact, not developers [31] and do not possess the technical skills necessary to deploy and maintain low-level system software, such as Linux and the Apache web server. Iivari, et al. note that the "user population of OSS is becoming larger, including a growing number of non-technical users, who are not interested in OSS development, but only in the resulting solution" [16].

Many research studies of OSS organizational structures depict users and developers within the same project organizational hierarchy [7]. This hierarchy is commonly represented in an "onion" diagram, with active developers in the innermost layer and passive users on the outermost layer [18]. *Passive users* are not concerned with how an OSS project is developed, but rather are simply users of the final product [16]. Such users are not involved in the development or organizational structure of an OSS project: they do not contribute code, generate bug reports, or otherwise interact with the developers [14]. In 2006, Nichols and Twidale found that up to 90 percent of OSS users are passive users [31]. Passive users should not be in the organizational onion, or considered part of OSS development—rather they should be considered as the purpose for the developmental organization (i.e., the reason for the onion). These users should be viewed as the *consumers* of this metaphorical onion, that is, as the intended audience of OSS products.

The failure to adequately distinguish between users and developers may have its roots in another of Raymond's statements: "Every good work of software starts by scratching a developer's personal itch" [35]. This seemingly intuitive proposition begs a fundamental question: for whom are the developer's developing or innovating? For themselves, other developers, passive users, or all the above?

In truth, the notion that innovation follows a personal "itch" extends beyond OSS. Eric von Hippel draws an example from windsurfing. Pioneers in high-performance windsurfing in the 1970's began to customize their boards in order to jump higher in the air. These improvements led to general innovations in board design that did not originate with manufacturers, but rather with the windsurfers themselves. These innovations nevertheless found their way into future product lines [13]. These innovators or "lead users," improved windsurfing by satisfying their own needs. Similarly, many proponents of OSS believe, as does Raymond, that innovation in software begins with software developers (i.e., lead users) seeking to satisfy their own desires.

Such a narrow view of innovation fails to see the potentially motivating role that non-developer, non-technical users can play in the development of software. Continuing with von Hippel's example, manufacturers of windsurfing boards (not the lead users) incorporated the user-innovative improvements into products that benefited all windsurfers (in particular, the vast majority that contributed no innovations to the field of board design). While a good work of software may begin with an itch, someone must take this initial work and adapt it for general use. If OSS developers do not fulfill the role of the windsurfing manufacturer then who will develop for the average user?

1.2 Reflexive and Irreflexive Software Projects

Borrowing from the literature in sociology, we use the term *reflexivity* [2] to signify the intent of developing software for the benefit of oneself or others like oneself; and we use the term *irreflexivity* to mean the intent of developing software for the benefit of oneself and others not like oneself (i.e., passive users). We focus on developer intent rather than end-user usage. Reflexive activity is self-referential activity, that is, action which primarily affects and benefits the doer. If we apply this to open source software creation, then reflexivity is creating software which targets other developers, system administrators, or others in the field of information technology. Irreflexivity in open source software development is creating software for end users—users which are not limited to developers.

We explore reflexivity in software development through a discussion of both reflexive and irreflexive projects. Further, we examine the intended audiences of selected OSS projects, measuring the extent of reflexivity in OSS software development. We further assess the relative success of OSS projects when examined from the perspective of reflexivity.

1.3 Thesis

Using the SourceForge OSS project repository, we show that: (1) both reflexive and irreflexive projects exist; (2) reflexive projects are considerably more common than irreflexive projects;

and (3) the percentage of projects that can reasonably be identified as successful is significantly greater among the reflexive category than among the irreflexive category. Thus we intend to demonstrate that the majority of successful SourceForge projects are designed for users that are predominantly developers.

1.4 Project Description

We show that reflexive and irreflexive projects exist within open source software (OSS) by studying SourceForge projects. We discover the distribution of reflexivity within SourceForge and correlate reflexivity with success. We start by seeking answers to the following three questions:

1. How can we mine project metadata on SourceForge?
2. How can we classify SourceForge projects into reflexive and irreflexive categories?
3. How can we categorize success of SourceForge projects?

We divide our approach into two tasks: first, we seek to understand how projects are built and stored within SourceForge and determine which metadata is relevant to achieving our aims; second, we use this data to classify projects as reflexive or irreflexive.

1.4.1 Preliminary Study of SourceForge

We conducted a preliminary study of projects within the SourceForge repository, examining how projects are created on SourceForge and seeking to understand the options available to developers when creating a project. Using this information, we examined SourceForge's database tables to determine how it stores this data.

To answer the question of how to data mine project metadata, we analyzed the SourceForge Research Data Archive (SRDA). The SRDA allows researchers access to certain shared SourceForge database information such as a project name, description, supported operating system(s), intended audience(s), and topic(s). SourceForge is a well-known forge containing a large and diverse number of OSS projects. Consequently, it is an ideal choice for examining both developer-facing and end-user facing projects as there is no discrimination against any particular project type.

We created distributions of the metadata to gain insights into which metadata is meaningful to our study. In addition, SourceForge makes available a statistics API whereby the number of project downloads can be obtained. Using this API we were able to obtain the number of downloads of all SourceForge projects.

Our initial approach was to evaluate projects to determine if they were designed for a general end-user audience–that is, irreflexive. However, we determined inferring irreflexivity based solely on a project's metadata very difficult to achieve; we found inferring reflexivity easier to accomplish. We developed four criteria to categorize projects as reflexive based on three metadata categories useful in determining reflexivity: supported operating system, intended audience, and a project's SourceForge topic. Such inference is made knowing that our categorizing is not based on actual usage. It may be possible that developers intend their software project to be reflexive (for other developers) but actual usage is irreflexive.

This preliminary study resulted in a paper titled "Reflexivity, Raymond, and the Success of Open Source Software Development: A SourceForge Empirical Study." Chapter 2 is the full version of the paper which was published in the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE, April, 2013) in Recife, Brazil [9].

1.4.2 Empirical Study of SourceForge

For our next study, we extended on our previous work by creating a statistical model using a random sampling of SourceForge projects. This improved on our previous study in three ways:

- We included more projects in our dataset (370,404 compared to 211,654).
- We included an analysis of how reflexivity has changed within SourceForge over time.
- We manually examined project information to classify projects as reflexive or irreflexive.

- We employed multiple reviewers to measure our rating consistency and accuracy.

Our previous work relied heavily on project metadata. Consequently, we could not classify projects that were missing metadata; metadata such as intended audiences, supporting operating systems, or relevant topics. Our automated approach found that 143,310 (68%) of the 211,654 SourceForge projects with metadata were reflexive.

Building on our previous work, we again reduced Raymond's statement ("Every good work of software starts by scratching a developer's personal itch") into testable hypotheses and added a test to determine whether the number of irreflexive projects has increased over time.

We explored two approaches to answer our hypotheses: machine learning and regression analysis. Machine learning is concerned with producing general hypotheses for making predictions [20]. Our hypotheses focus on measuring the number of reflexive and irreflexive projects within SourceForge and correlating this number to success (number of downloads). We examine open source projects as they now exist rather than making predictions about future projects—our approach is descriptive rather than predictive. Creating a prediction tool is outside the scope of our analysis.

We decided that regression analysis provided a way to correlate success in open source (as measured by the number of times a project is downloaded) and reflexivity. "Regression analysis is used to describe the distribution of values of one variable, the *response*, as a function of other—*explanatory*—variables" [33, p. 178]. We use success as our response variable and reflexivity as the explanatory variable. Using a random sampling process of selecting SourceForge projects, we draw inferences to the entire population of SourceForge projects.

Our findings suggest that the majority of SourceForge projects do appear intended primarily for technical users. Our findings further suggest that reflexive projects tend to be more successful than those intended for a more general audience. It is critical to note that our findings are only correlations; causality is not addressed in this study. We do not yet know whether reflexivity drives project success, or whether success drives an increase in the reflexivity expressed by developers through project indicators.

This paper is ready for submission, but has not yet been published. The full version of this paper is included as Chapter 3 of this thesis.

1.5 Related Work

1.5.1 Usability

Nichols and Twidale set up a usability framework to empirically study the issue of OSS usability. In 2003, they examined the differences between developers and users [30]. In 2005, they studied bug reports of several OSS projects, exploring ways developers categorize users [31]. In 2006, they analyzed usability (and the lack of technical expertise on the part of users) in OSS projects [42].

Netta Iivari examined an end-user, media-player project on SourceForge [14]. Within this project, developers struggled to understand the final end-user of their software project. Moreover, their failure to properly identify their intended audience manifested itself in the terms they used when speaking of users. They spoke about *typical users*, *novice users*, *non-technical users*, and *users in general*—struggling to agree on which term properly identified their intended user. This confusion of language only highlights the lack of understanding on how best to address user-related issues.

Singh et al. highlight the difficulties created by the so-called "curse of knowledge": informed developers may have problems understanding the problems of those with less skill than themselves [40]. Accordingly, even if a project is developed and intended for use by less technically savvy users, the resulting lack of a well-designed interface can inhibit its usability and subsequent appeal to its intended audience.

Rantalainen et al. claim that most open source projects are not designed for a general non-technical audience [34]. A typical user has neither the technical skill nor desire to use the vast majority of infrastructure and tool-type open source projects.

While the body of work identifying the lack of usability in OSS projects is growing, we find no large scale user-centered analysis of a large OSS repository, such as the one we propose here.

1.5.2 Success of Open Source Software

In general, success can be thought of as the degree to which a solution meets human goals or needs. Of course, directly quantifying human goals and needs is difficult. For commercial software, success is often measured by the profit generated from software sales. However, revenue models are very different in OSS contexts, since the source code is publicly released. Also such data would be difficult and time consuming to obtain for a large number of projects, not to mention that for most SourceForge projects, financial data are simply not available. Consequently, we consider project metrics that are readily available in the SourceForge context—release frequency, developer activity, software quality, and downloads. Each of these metrics, of course, presents a different perspective on project success, but all likely correlate to some degree with the meeting of human goals and needs.

Success of OSS projects has been measured at various levels of analysis, including internal and external factors. Internal analysis examines factors relating to the nature of the OSS development process, such as growth in the size of the developer community [22], developer contribution [27], and activity level within open source projects [21] [32]. External analysis examines the products of OSS development, such as project popularity [44] and project usage/adoption (such as Netcraft's survey of Apache web server market share[1], or—as will be used within our study—the number of times a project is downloaded).

Open source artifacts such as source code, mailing lists, and commit logs can be used in an internal analysis. Internal success can be measured by the motivation of developers [24] or ideology within the open source community [35]. Some research has focused on organizational structure concluding that an open source development process can produce high quality and

[1] Netcraft - Web server survey archives 2006.

widely distributed software [46]. Linux, Apache Server, and Eclipse are often cited as proof of OSS development success.

1.5.3 SourceForge

SourceForge has been the subject of many previous studies. In 2002, Krishnamurthy analyzed SourceForge by counting the number of projects within six product development categories. He concluded that most mature projects were developed by a small number of developers [21]. In 2005, Rainer and Gale examined the prevalence of SourceForge projects under active development (less than 1%) [32]. In 2005, Weiss [45] showed ways that SourceForge organizes and represents data about projects. Building on these studies, English and Schweik classified SourceForge projects by release and developer activities [6]. Each of these papers informed us in our study of categorizing SourceForge projects.

1.6 Threats to Validity

In our first study we identified three threats to the validity of our study. We describe each of these threats and how we dealt with them in our second study.

First, we examined fewer projects in our original study 211,654 projects compared to 370,404 in this study. One reason for the smaller dataset was that we excluded projects with missing metadata necessary to categorize projects as reflexive. We recommended changing the methodology to include projects with missing metadata. In this study, we did not exclude projects with missing metadata. We found many projects with missing project descriptions or projects with descriptions that lacked sufficient details for us to categorize as reflexive or irreflexive. In the original study, we also excluded .u projects (as described in Section 3.4). We investigated further and concluded these are auto-generated profile pages created when users register an account on SourceForge.

Second, we identified cases of project binaries that are bundled and distributed with other software (e.g., some open source utilities are distributed with certain versions of Linux).

Our download metric failed to capture all instances where a project is used, hence potentially under-measuring project success. Dealing with this threat would require determining which projects are bundled elsewhere. Given the time and resources available to us, such an undertaking is outside the scope of our study. Another shortcoming was in using the term "download," which insinuates that users are intentionally downloading an OSS project, which may not be true. Again, obtaining the intentions of users is outside the scope of this study.

Using downloads as a raw count ignores the fact that some very successful projects may apply only to a small niche, with relatively high success reflected in a low number of downloads. We accounted for this by placing projects into buckets and measuring the proportion of reflexive and irreflexive projects within each bucket. This permits correlating success with reflexivity by comparing projects with similar download counts.

Third, our classification of SourceForge projects is a subjective judgment and we recommended refining our classification methodology through the use of multiple reviewers. We identified a potential for bias in finding reflexivity because we were looking for it, a kind of *self-fulfilling prophecy*. To account for this we employed the help of external judges to rate projects and compare their ratings to ours. We find consistency between the judges and correlation between the ratings of the judges and our ratings.

Lastly, in our first study we used *inclusion* criteria to classify projects with respect to reflexivity. We noted that searching for only positive cases may inflate the results. We recommended using both inclusionary and exclusionary criteria. To account for this, we developed a nominal rating scale that included definitions of both irreflexivity and reflexivity. Raters were allowed to rate projects as reflexive, irreflexive or both. This allowed us to test for both irreflexivity and reflexivity. While we will report on total number of downloads, correlating this to success may not be warranted.

1.7 Going Forward

Madey, et al., argue that "The OSS movement is a phenomenon that challenges many traditional theories in economics, software engineering, business strategy, and IT management" [26]. We concur that OSS is a phenomenon—a phenomenon worthy of study. This thesis provides insight into two types of open source projects: developer-facing (reflexive projects) and end-user facing (irreflexive projects). We find a correlation between reflexive projects and how often a project is downloaded. We find that most OSS projects are designed for developers, system administrators, and others in the field of information technology. Moreover, we find an increase in the number of irreflexive projects from 1999 through 2001.

We speculate that for OS software to achieve greater success (i.e., more users choosing OSS over proprietary) will require OSS developers to focus more on usability—more concern for the passive user.

Chapter 2

Reflexivity, Raymond, and the Success of Open Source Software Development: A SourceForge Empirical Study

2.1 Introduction

Open source evangelists, such as Eric Raymond, have repeatedly stressed the critical role of users in open source software (OSS) development. For example, Raymond's articulation of "Linus's Law," that "given enough eyeballs, all bugs are shallow," reinforces the essential role that users play in OSS development [35, p. 29]. However, in *The Cathedral and the Bazaar* [35] Raymond consistently conflates users and developers; indeed in the early days of OSS development most users were, in fact, OSS developers. Contemporary OSS research typically fails to adequately distinguish between OSS developers and OSS users [46].

In fact, most OSS users today are *not* developers [31] and do not possess the technical skills necessary to develop software. Iivari, et al. note that the "user population of OSS is becoming larger, including a growing number of non-technical users, who are not interested in OSS development, but only in the resulting solution" [16]. In 2006, Nichols and Twidale found that up to 90 percent of OSS users are *passive users* [31] who are not interested in, or perhaps even capable of, writing OSS software.

In this paper we empirically test Raymond's claims concerning OSS developers/users and, in particular, the impact of their relationship on project success. Raymond states, "Every good work of software starts by scratching a developer's personal itch" [35, p. 25]. We are interested in the degree to which this statement holds, given that most OSS users are not developers.

Empirically testing Raymond's statement requires precise terminology. The term *reflexivity* is defined in sociology as "the capacity of language and of thought. . . to turn or bend back upon itself, to become an object to itself" [2, p. 2]—that is, a reflexive action is one of self-reference, primarily intended to affect or benefit the doer. We adapt the term to the context of OSS as follows:

- **Reflexivity** – *the intent of developing software for the benefit of oneself or others like oneself* (i.e., for other developers).
- **Irreflexivity** – *the intent of developing software for the benefit of others not like oneself* (i.e., for passive users).

Note that we focus on developer intent, rather than on end-user composition for two reasons: 1) Intent is easier to measure from available project artifacts; and 2) Consistent with Raymond's statement, developer intent drives motivation to participate in OSS projects. Thus reflexive software projects are primarily designed for developers, system administrators, or others in the field of information technology (rather than casual users). Irreflexive projects are designed primarily for passive users, which may coincidentally include developers. Figure 2.1 shows the reflexive and irreflexive relationships between developers and users.

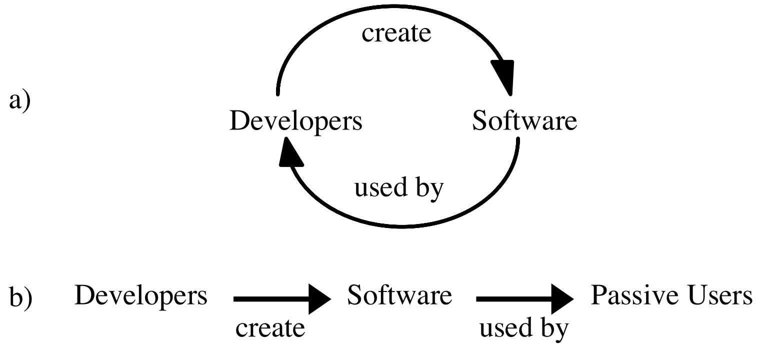


Figure 2.1: a) Reflexive and b) irreflexive software processes

We now reduce Raymond's statement that "Every *good* work of software starts by scratching a developer's *personal* itch" [35, p. 25, emphasis added], into two testable hypotheses:

1. Most OSS projects are reflexive (i.e., "…scratching a developer's personal itch").
2. Reflexive projects are more successful (i.e., "Every good work of software…").

Thus our goal in this paper is to explore the role of reflexivity within OSS development, and in particular, to test Raymond's assertions regarding the impact of reflexivity on project success. We explore these hypotheses through an empirical study of SourceForge, a popular OSS repository.

In Section 2.2 we briefly describe our methods, including a description of our data source, methods for measuring success and reflexivity, and methods for data filtering. We then present results in Section 2.3, followed by threats to validity in Section 2.4, and general conclusions in Section 3.9.

2.2 Methods

In this section we discuss our data sources (Section 2.2.1), metrics for success and reflexivity (Section 2.2.2), and process for cleaning the data to obtain the final results (Section 2.2.3).

2.2.1 Data Source

SourceForge has been the subject of many previous empirical studies [21] [45] [6] [32], and is a well-known forge supporting a large and diverse set of OSS projects. It is a reasonable choice for exploring the reflexivity of software projects because it does not discriminate against the various types of projects that a developer may create.

The University of Notre Dame hosts the SourceForge Research Data Archive[1] (SRDA), which provides researchers with a wide array of SourceForge metadata, pre-extracted into a usable database schema. We used the September 2011 SRDA data dump, which includes all SourceForge projects created from the inception of SourceForge through September 30, 2011. From this source we obtained all metadata considered in this paper, with the exception of downloads. The SRDA currently does not maintain up-to-date download statistics. For

[1] http://zerlot.cse.nd.edu/mediawiki

this information, we utilized SourceForge's API,[2] from which we obtain download statistics for projects from January 1, 2000 through September 30, 2011 (i.e., covering the same time range as the SRDA data). Our definition of *download* as a metric is consistent with the definition employed by the SourceForge API, namely any file downloaded from a project's file release space, independent of the contents or type of the file.

2.2.2 Measuring Success & Reflexivity

In general, success can be thought of as the degree to which a solution meets human goals or needs. Of course, directly quantifying human goals and needs is difficult. For commercial software, success is often measured by the profit generated from software sales. However, revenue models are very different in OSS contexts, since the source code is publicly released. Also such data would be difficult and time consuming to obtain for a large number of projects, not to mention that for most SourceForge projects, financial data are simply not available. Consequently, we consider project metrics that are readily available in the SourceForge context, for example, release frequency, developer activity, software quality, and downloads. While each of these metrics presents a different perspective on project success, all likely correlate to some degree with the meeting of human goals and needs.

For the purposes of this study we use *downloads* to represent success. Despite its limitations (which we discuss in Section 2.4), we believe the number of downloads is a reasonable metric, particularly since downloads directly reflect user demand, and one would expect demand to significantly correlate with meeting human goals and needs. Further, if Raymond is right, one would expect project reflexivity to positively impact downloads. Therefore, we examine whether even minimal evidence exists for Raymond's claims.

In order to measure reflexivity, we identified three types of relevant metadata from SourceForge projects: intended audiences, relevant topics, and supported operating systems. For these attributes, SourceForge provides predefined lists, from which project creators may

[2] http://sourceforge.net/p/forge/documentation/Download%20Stats%20API/

select zero or more of the possible choices. However, none of the metadata we need for assessing reflexivity is required to create a project (and in fact, many SourceForge projects are missing the needed data, which we discuss in Section 2.2.3).

The first classification step is to identify which types of metadata responses are indicative of reflexivity. We tag each of the metadata options as either indicative or not indicative of reflexivity, according to the following rules:

- *Intended audiences* – This category is our strongest indicator of potential reflexivity, since developers directly select these classifications to characterize their audience; meaning is ascribed by the project creator, rather than by researchers, as is the case with the other two criteria.[3] We label three of the 25 intended audience roles as strongly suggesting reflexivity: developer, system administrator, and quality engineer.
- *Relevant Topics* – Topics are listed in a tree, with first-level nodes covering broad categories such as *System* or *Multimedia*, and leaf nodes being more specific (e.g., *Formats and Protocols::Data Formats::JSON*). In total, 373 topics are provided. SourceForge uses these topics for indexing projects, and thus they play an important role in guiding potential users to new projects. Some root topics are clearly descriptive of software infrastructure (e.g., Software Development). These topics and their subtopics are tagged as indicative of reflexivity. Other root topics are clearly irreflexive in nature (e.g., Games/Entertainment). For verification, we manually examine all subtopics to confirm they do not contradict the root-level tag. For ambiguous nodes (e.g., Communications) we make a best guess.[4]

[3] Why not use intended audience exclusively? Because developers themselves may not be entirely cognizant of their actual audience. For example, they may indicate a general audience while creating software for other developers. This masking effect would be particularly likely if Raymond's treatment of users as developers is representative of the OSS developer mindset.

[4] Obviously some subjectivity is required in order to translate project metadata into a declaration of reflexivity. For this preliminary study we follow intuition where no obvious choice is apparent, and as such, some readers may disagree with certain designations. Future work is required to develop a more robust process (see Section 2.4).

- *Supported Operating Systems* – SourceForge presents a list of 83 supported operating system choices. We classify operating systems based on the following question: *If this were the only supported operating system, would this choice imply that the project is intended primarily for developers, system administrators, or other information technology professionals?* Linux, for example, would be labeled as somewhat indicative of reflexivity.

Table 2.1 lists all metadata values we label as indicative of reflexivity. As discussed above, Intended Audience provides the most evidence of reflexivity, though relevant topics and supported operating systems may additionally suggest reflexivity. Accordingly, we apply the latter categories with more caution. Reflexive projects are identified as those meeting at least one of the following criteria (see also Table 2.2): the project lists *any* of the three intended audiences identified as indicative of reflexivity—regardless of who else is identified as well (criterion C1); OR the project lists *only* topics thought to imply reflexivity (criterion C2); OR the project lists *only* operating systems thought to imply reflexivity (criterion C3). The classification process is more complex for projects that have a mixture of reflexive and irreflexive operating systems and/or topics. We deal with these cases by creating an additional classification rule: a project is reflexive if it contains at least one likely reflexive operating system AND at least one likely reflexive topic (criterion C4).

2.2.3 Data Cleaning

As discussed previously, we focus on three indicators of reflexivity: 1) intended audiences, 2) relevant topics, and 3) supported operating systems. Projects that do not specify at least one of these indicators cannot be classified. In addition, projects for which download information is unavailable cannot be used. In this section we discuss project exclusions and their potential impact on the results.

Of the 457,288 projects accessible via the SRDA, 245,059 projects (54%) are missing all three reflexivity indicators and are, therefore, excluded. Our preliminary examination

Intended Audiences	developer, system administrator, enduser_qa
Relevant Topics	Communications::Chat::Unix Talk, Communications::Email, Database (all), Formats and Protocols (all), Internet (all except Browsers), Office/Business::Enterprise:: Data Warehousing, Security::Cryptography, Software Development (all), System (all), Terminals (all), Text Editors (all)
Supported Operating Systems	aix, amigaos, beos, brew, bsd, bsdos, cygwin, dosemu, ecos, emx, fink, freebsd, gnuhurd, hpux, irix, limo, linksyswrt54g, linux, mingw_msys, mswin_nt, mswin_server2003, netbsd, openbsd, openvms, os_projectkernel, qnx, sco, uclinux, vxworks, winnt

Table 2.1: Metadata values labeled as indicative of *reflexivity* ("all" refers to inclusion of child topics)

Criterion	Rule
C1	Project lists at least one intended audience considered indicative of reflexivity.
C2	Project lists only reflexive topics.
C3	Project lists only reflexive operating systems.
C4	Project lists at least one reflexive operating system AND at least one reflexive topic.

Table 2.2: Classification criteria (if at least one criterion is met, the project is classified as *reflexive*)

of these excluded projects suggests a consistency with the report by Healy and Schussman of "spectacularly skewed" distributions for most project indicators in SourceForge data (e.g., developers, commits, downloads, etc.) [12]. Our operational assumption is that these excluded projects are, on the whole, neglected and/or abandoned projects, and that such exclusion is consistent with the goals of this study. Further, given the importance of metadata to the distribution of SourceForge projects (i.e., helping users find and utilize projects), it seems unlikely that active projects would neglect to provide these data. As further support, we observe that 219,252 (89%) of the excluded projects provide no metadata whatsoever, while 146,787 (60%) are auto-generated projects (created by SourceForge in response to user account creation). Further investigation is required to verify the inactivity assumption for *all* excluded projects.

Of the 212,229 projects that appear active, a small number (575, or 0.3%) return a 404 not found error by SourceForge's API. We know that SourceForge allows some projects to keep download statistics private, and this is a possible explanation for at least some of the missing data. However, we attempted to look up some of these projects (approximately 15) on the SourceForge website and found that most (approximately 13) did not exist. According to English and Schweik, SourceForge "occasionally purges defunct projects" [6, p. 4]. Further work is required to confirm these assumptions. This last pruning leaves us with 211,654 projects for which we have sufficient data to address our hypotheses.

2.3 Results

In this section we present results. In Section 2.3.1 we examine project distributions across the three reflexivity indicators previously shown in Table 2.1. In Section 2.3.2, we examine project distributions across the four reflexivity criteria from Table 2.2. We also address our first hypothesis (H1), regarding whether OSS projects tend to be reflexive (as Raymond insinuated). Finally, in Section 2.3.3, we examine project distributions by downloads and address our second hypothesis (H2), regarding whether reflexive OSS projects tend to be more successful than irreflexive projects.

2.3.1 Reflexivity Indicators

In this section, we examine project distributions across the three reflexivity indicators shown in Table 2.1. Note that developers may select zero or more predefined responses for each category. Thus the sum of the numbers in each chart do not necessarily equal the total number of projects in the dataset. In Figures 2.2-2.4, metadata selections are listed on vertical axes, project counts for each selection are represented by horizontal axes, grey bars represent all projects, and black bars represent the subset of projects matching the chart's corresponding reflexivity criterion.

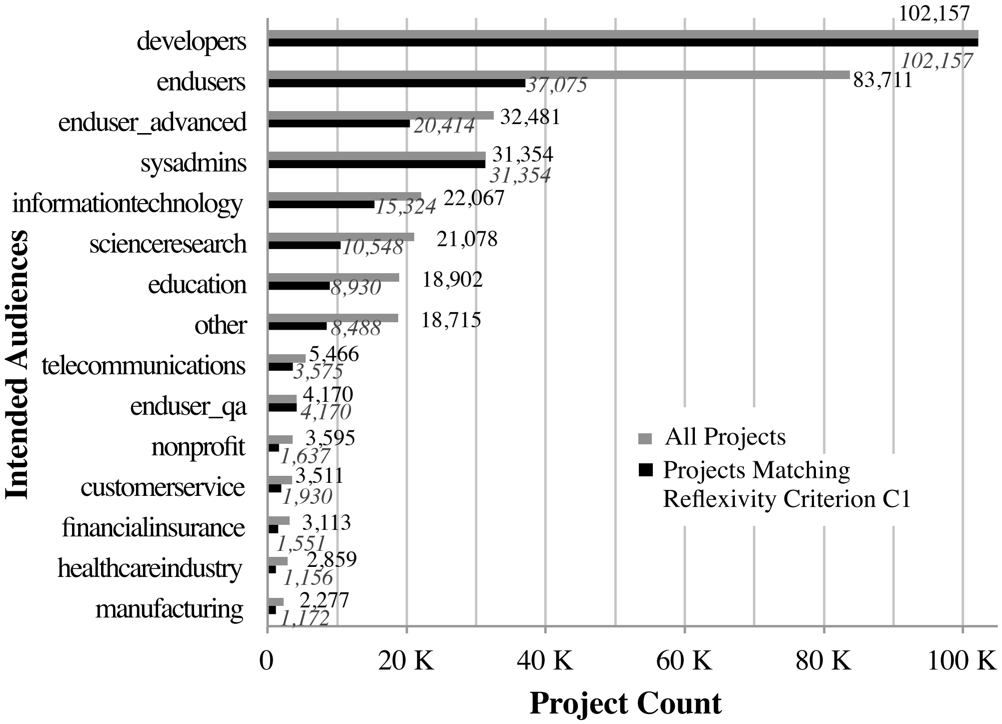


Figure 2.2: Distribution of projects by intended audiences (top 15 of 25 shown, accounting for over 97% of the selections)

Figure 2.2 shows the distribution of projects across the top 15 intended audiences (the remaining 10 account for less than 3% of the selections). Any project specifying at least one of three specific intended audiences—`developers`, `sysadmins`, and `enduser_qa`—is determined to be reflexive. Notice that the top two categories, `developers` and `endusers`, comprise more than 50% of all selections, one of which directly implies reflexivity, whereas the other seems to imply irreflexivity. The `developers` selection is more common, though, so Raymond's claim (H1) is supported. Further, `endusers` is a somewhat general category, which could also include developers. This bias is against reflexivity; thus more precise data for this category would likely lead to an increase in reflexivity. In fact, almost half of the `endusers` selections belong to projects that also specify one of the three reflexive audiences (i.e., projects matching reflexivity criterion C1).

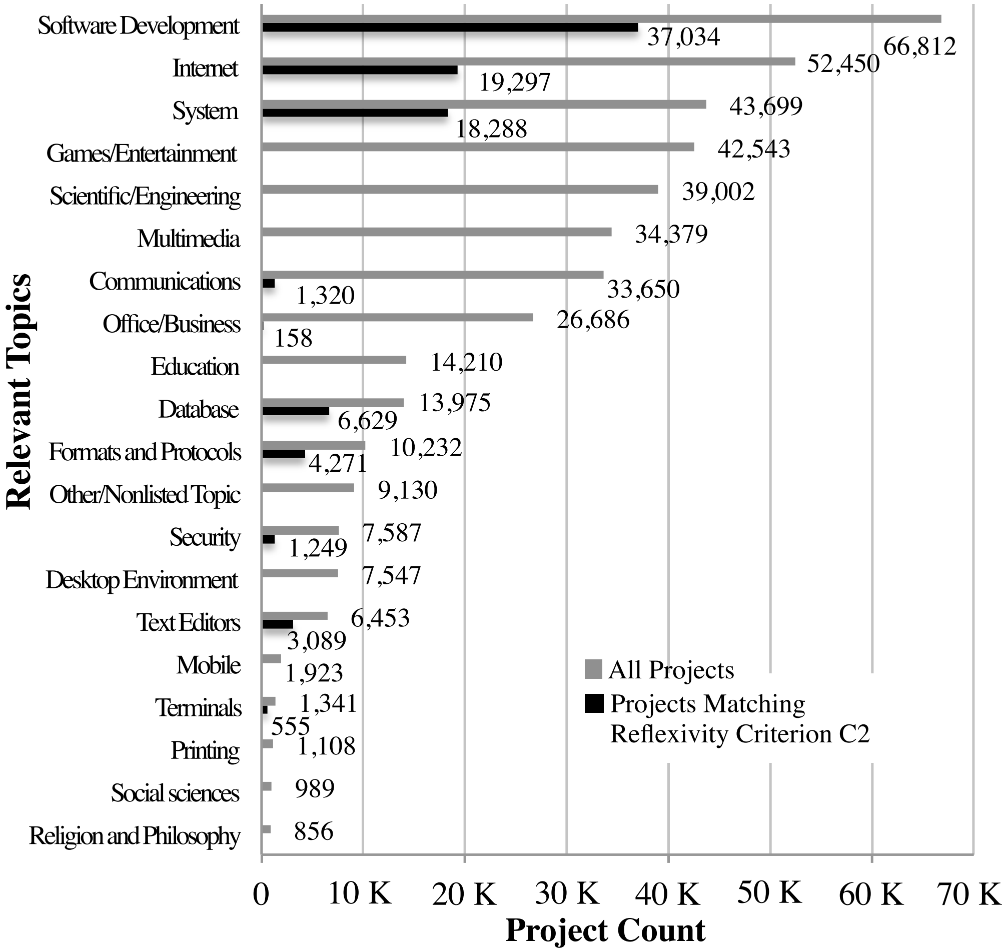


Figure 2.3: Distribution of projects by relevant topics (all 20 *root* topics shown)

Figure 2.3 shows the distribution of projects across topics. Note that SourceForge organizes topics in a tree structure; for brevity, we only show root topics. The top three topics—`Software Development`, `Internet`, and `System`—all seem indicative of reflexivity, and in fact, many of the projects specifying these topics match reflexivity criterion C2. Further, the prevalence of `Software Development` and `System`, in particular, seem to indicate that SourceForge hosts a large number of software “plumbing” and tooling projects.

Figure 2.4 shows the distribution of projects across the top 15 supported operating systems (the remaining 68 account for less than 10% of the selections). Interestingly, in contrast to topics, only three of the operating systems categorized as implying reflexivity show up in the top 15. From this perspective, therefore, it would seem that Raymond was

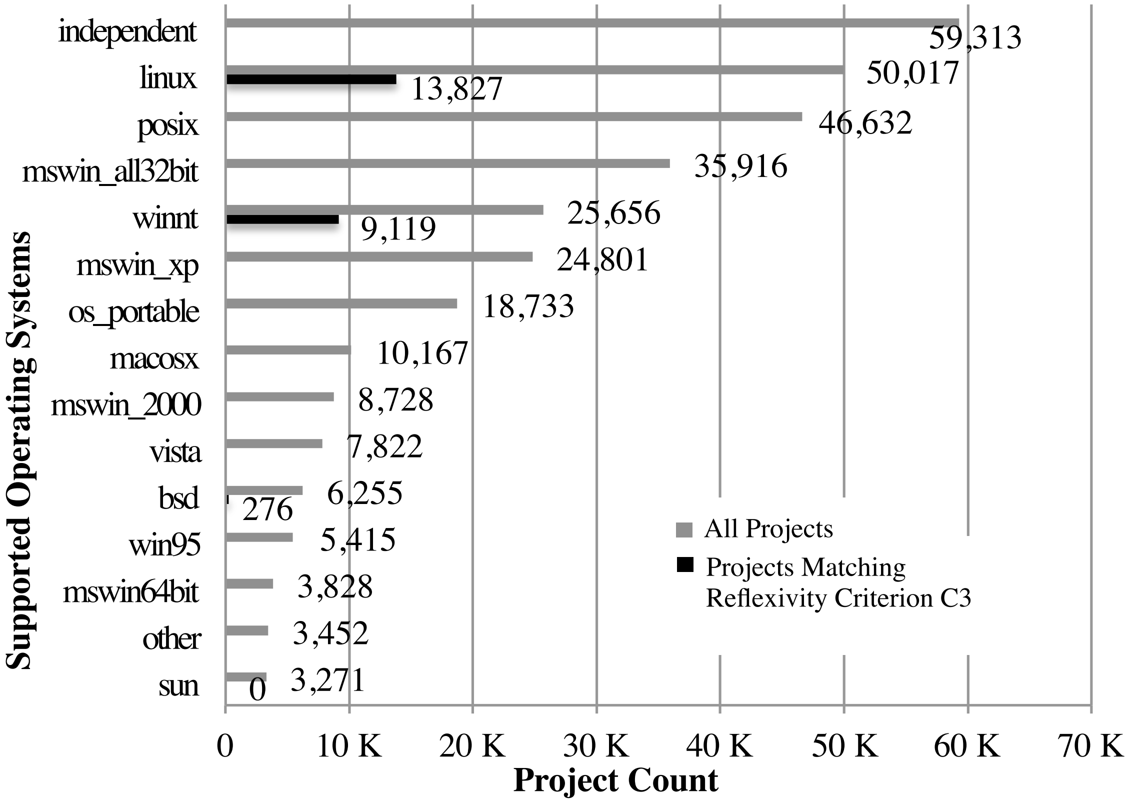


Figure 2.4: Distribution of projects by supported operating systems (top 15 of 83 shown, accounting for over 90% of the selections)

wrong. However, just because some operating systems are rarely used by non-technical users does not mean that more popular operating systems (e.g., Windows) do not also include many developers as users.

2.3.2 Prevalence of Reflexivity

Table 2.3 shows the results of applying our reflexivity criteria (from Table 2.2) to all 211,654 projects in the final dataset. Note that the values in each column do not sum to match the numbers listed in the last row. This is because any given project may match multiple criteria.

It is clear from the table that criterion C1 (intended audiences) matches far more projects than any other criterion. Almost half of the projects (51%) are declared reflexive due to this criterion. Overall, 143,310 projects (68%) match at least one of the four criteria. Thus given our process for assessing reflexivity, Raymond's claim (H1) is supported. It does

Reflexivity Criterion	Matching Projects	Percent Matching
C1 (IA)	108,077	51%
C2 (T)	58,580	28%
C3 (OS)	24,407	12%
C4 (T and OS)	43,059	20%
$C1 \cup C2 \cup C3 \cup C4$	143,310	68%

(IA) Intended Audiences, (T) Relevant Topics,
(OS) Supported Operating Systems

Table 2.3: Counts and percentages of projects matching the reflexivity criteria (out of 211,654 projects)

appear that reflexive projects are more common than irreflexive projects (although 68% is not an overwhelming majority). It would be interesting to calculate changes in reflexivity as a function of time. We would expect to find that reflexivity has become less common than it was at the time Raymond made his statement. We leave this assessment to future work.

2.3.3 Success of Reflexive Projects

The number of downloads per project spans a large range, from zero to well over six hundred million. The distribution is also extremely skewed. We therefore use a log_{10} scale in Figure 2.6 and Tables 2.4 and 2.5. Note that 48% of all projects have never been downloaded. Consequently, these projects have a significant pull on summary statistics and histograms. We therefore exclude them from Figure 2.5. Note also that while the log operation is undefined at zero, in Figure 2.6 we include projects with zero downloads for graphical comparison.

	Mean		Median		Std.Dev.	
Reflexive	38,764	(4.59)	68	(1.83)	2,542,073	(6.41)
Irreflexive	14,403	(4.16)	0	(-)	587,595	(5.77)
Combined	30,898	(4.49)	18	(1.26)	2,118,275	(6.33)

Table 2.4: Download statistics for all projects (log_{10} scale in parentheses)

Table 2.4 presents download statistics for all 211,654 projects in the final dataset (including projects with zero downloads). For the 102,553 projects that have never been

	Mean		Median		Std.Dev.	
Reflexive	71,102	(4.85)	738	(2.87)	3,442,476	(6.54)
Irreflexive	31,786	(4.50)	497	(2.70)	872,592	(5.94)
Combined	59,942	(4.78)	664	(2.82)	2,950,114	(6.47)

Table 2.5: Download statistics for projects exceeding zero downloads (log_{10} scale in parentheses)

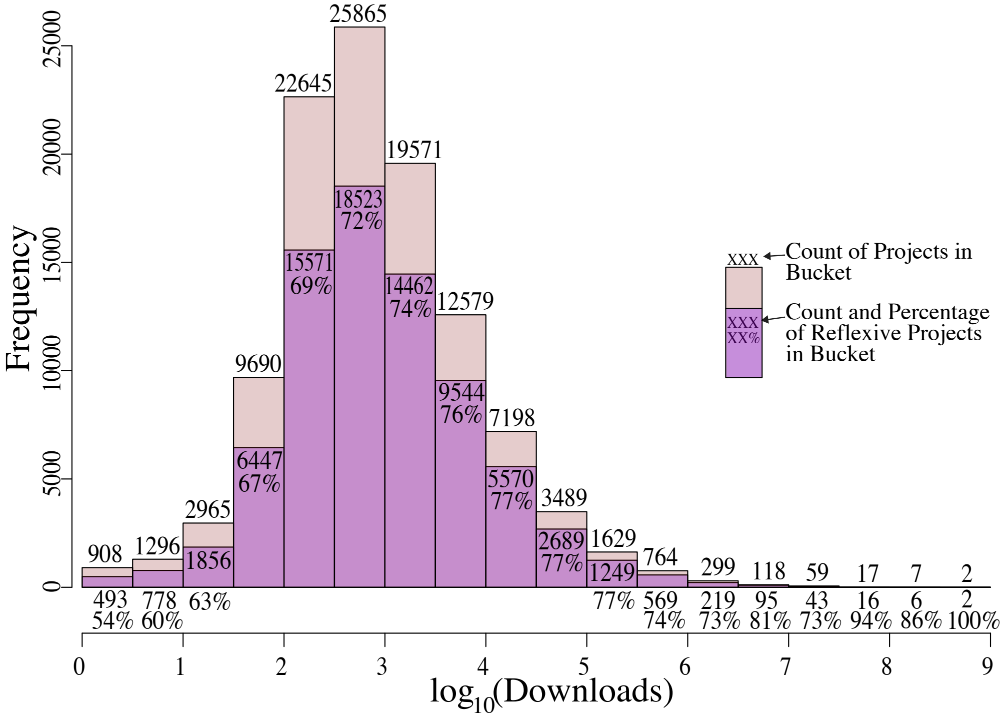


Figure 2.5: Distribution of projects by downloads (log_{10} scale), total projects overlaid with reflexive projects (projects with zero downloads excluded)

downloaded, 65,178 (64%) are reflexive. These projects significantly pull down the means and medians for projects with greater than zero downloads (as seen in Table 2.5). However, with or without these “unsuccessful” projects, our conclusions are the same. Clearly, the mean (and median) number of downloads is significantly larger for reflexive projects than for irreflexive projects. Thus Raymond’s claim that reflexive projects tend to be more successful than irreflexive projects (H2) is supported.

Figure 2.5 provides a histogram of projects by downloads (excluding those with zero downloads, as mentioned previously). The histogram reinforces our conclusion that the majority of projects are reflexive (H1). It also demonstrates that the preponderance of

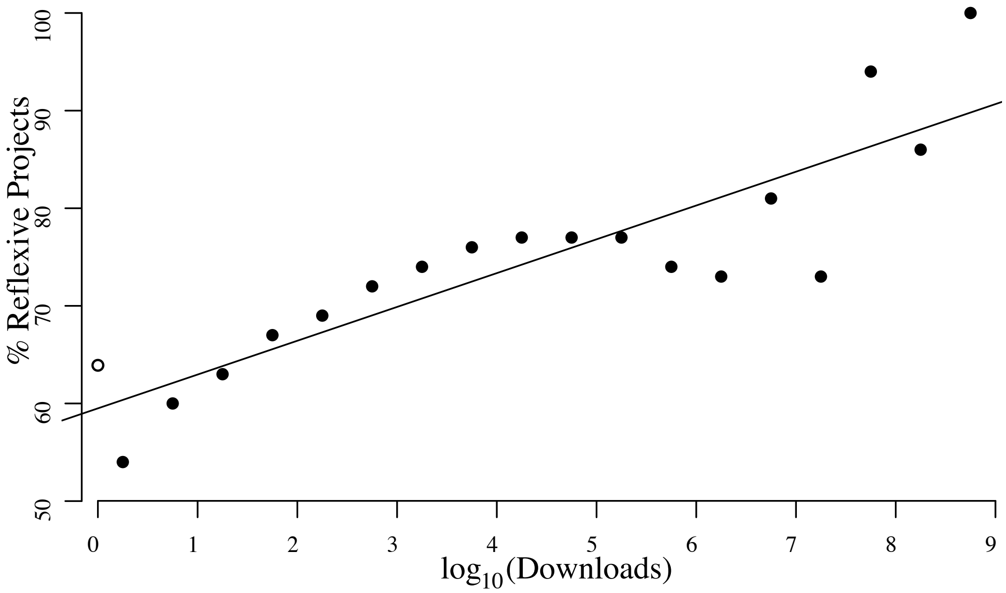


Figure 2.6: Percentage of reflexive projects bucketed by download count (log_{10} scale) with best-fit line (buckets match those in Figure 2.5; for graphical comparison, projects with zero downloads are included; log of zero is undefined, so an open circle is used)

reflexive projects holds generally across the full range of the data (i.e., failed projects are not driving the measured reflexivity). We also observe (not unexpectedly) a significant power law distribution with very successful projects being relatively rare, but accounting for most of the total downloads from SourceForge. In fact, the 502 projects boasting more than 1,000,000 downloads—less than 0.24% of all projects in the final dataset—account for more than 82% of all downloads.

The percentages of reflexive projects (corresponding to buckets in the histogram) are graphed in Figure 2.6. Notice that these percentages positively correlate with downloads, thus strongly supporting Raymond's claim (H2). This trend is stable throughout the main body of the data. Only where the data become sparse do we see a dip in the percentage, followed by significant (though overall upward) fluctuations. Assuming reflexivity is accurately classified, this trend appears robust.

2.4 Threats To Validity

Since this research is still at an early stage, it is subject to several important threats to validity. First, we exclude projects for which the available data are insufficient (see Section 2.2.3). The vast majority of these exclusions appear to be unused projects (many of which were auto-generated). However, some may be important, particularly the 575 for which we have reflexivity indicators, but no download statistics. Until we have investigated these projects further, we cannot determine how significantly they impact the results. However, unless many of them exceed one million downloads (thus falling into buckets where the data are sparse), they are unlikely to outweigh the observed results.

Second, the downloads metric fails to deal with the fact that some project binaries are bundled and distributed with other software (e.g., some open source utilities are distributed with certain versions of Linux). In this case, the downloads metric fails to capture all instances where a project is used, hence potentially under-measuring project success. Another shortcoming of the downloads metric lies in the fact that the term "downloads" insinuates that users are intentionally downloading an OSS project, which may not be true. For instance, some libraries and utilities are downloaded every time an operating system is installed (e.g., the project *corefonts* is downloaded automatically from SourceForge with many Linux installs). Finally, using downloads as a raw count ignores the fact that some very successful projects may apply only to a small niche, with relatively high success reflected in a low number of downloads. How we deal with this nuance depends largely on our definition of success. In general, readers should recognize that our use of downloads assumes a specific definition.

Third, our classification of SourceForge projects as reflexive requires up-to-date information from developers, as well as subjective judgment on our part to interpret that information. In general, we believe few developers forget to update supported operating systems, given the significant impact that such information has on product distribution. Further, it is reasonable to assume that the other two attributes do not change much once a project becomes well-established. Nevertheless, we have not yet validated these assumptions. As for subjective judgment, in future iterations of this research, we hope to refine our classification by having multiple researchers independently, manually examine a random sample of projects in depth (e.g., reading project websites).

Lastly, we use *inclusion* criteria to classify projects with respect to reflexivity. However, searching for only positive cases may inflate the results. If we were to use exclusion criteria instead, we may find fewer reflexive projects. Further work is needed to test for this bias (e.g., developing criteria designed to identify *ir*reflexive projects instead and comparing the results to those already obtained).

2.5 Conclusions

Open source users have traditionally been thought of as similar to (or even synonymous with) developers. While almost certainly reflecting a cultural reality in the dawn of open source software, that demographic appears to be shifting. Still, our desire to understand the fitness of open source developers to build software for an audience unlike themselves motivated our empirical analysis of the ratio of reflexivity among OSS projects (H1), and the relative success of projects as a function of reflexivity (H2).

We found that 68% of SourceForge projects are reflexive—thus supporting the first hypothesis (H1). Measuring success by project downloads, we also found that the prevalence of reflexivity is positively correlated with success—thus supporting the second hypothesis (H2). These findings suggest that the majority of SourceForge projects do appear intended primarily for technical users. These findings further suggest that reflexive projects tend to be more successful than those intended for a more general audience. It is critical to note that these findings are only correlations; causality is not addressed in this study. Even if causality can be shown to hold, we do not yet know whether reflexivity drives project success, or whether success drives an increase in the reflexivity expressed by developers through project indicators. We also do not know the effect that shifting OSS developer and user demographics may play on measured reflexivity ratios as well as project success. Put another way, early projects have had more time to be downloaded than successful projects made available more recently.

Chapter 3

Passive Users, Reflexive Developers, & Success in OSS Development: A SourceForge Empirical Study

> *The Linux OS is not developed for end users but rather, for other hackers. Similarly, the Apache web server is implicitly targeted at the largest, most savvy site operators, not the departmental intranet server... OSS development process[es] are far better at solving individual component issues than they are at solving integrative scenarios such as end-to-end ease of use.* – Microsoft internal memo [43]

3.1 Introduction

3.1.1 Open Source Users

In 1999, Raymond asserted that OSS users and developers are not so different from one another, and yet acknowledged that developers "tend to be poor at modeling the thought processes of the other 95% [i.e., non-developers] of the population well enough to write interfaces" [36]. Open source software usage has increased since 1999 and the "other 95% of the population" are increasingly choosing open source software over proprietary solutions.

Until recently, many academic researchers used the term "user" to refer to developers who use and develop open source software [19]. This pattern followed the conflated terminology originally employed by Raymond, in which he referred to OSS developers as "users." Since that time, open source projects have increased their reach to a broad audience that includes a significant percentage of non-developers. A growing body of researchers has identified the need for OSS projects to address the issue of usability [3] [15] [37] [1] [10].

In fact, most OSS users do not possess the technical skills necessary to develop software [31]. Iivari, et al. note that the “user population of OSS is becoming larger, including a growing number of non-technical users, who are not interested in OSS development, but only in the resulting solution” [16].

Many research studies of OSS organizational structures depict users and developers within the same project organizational hierarchy [7], commonly represented in an “onion” diagram (see Figure 3.1), with active developers in the innermost layer and passive users on the outermost layer [18]. *Passive users* are not concerned with how an OSS project is developed, but rather are simply users of the final product [16]. Such users are not involved in the development or organizational structure of an OSS project: they do not contribute code, generate bug reports, or otherwise interact with the developers [14]. In 2006, Nichols and Twidale found that up to 90 percent of OSS users are passive users [31].

We recommend removing passive users from the OSS ”organizational onion” and viewing passive users as the *consumers* of this metaphorical onion, that is, as the intended audience of OSS products.

In this paper we empirically test Raymond’s claims concerning OSS developers and users and, in particular, the impact of their relationship on project success. Raymond states, “Every good work of software starts by scratching a developer’s personal itch” [35, p. 25]. We are interested in the degree to which this statement holds, given that most OSS users are not developers. And if this statement was true when Raymond made it, we question the degree to which it holds today.

In Section 3.2 we discuss our previous work testing Raymond’s claim in the context of SourceForge projects, and articulate the expansion on that work represented by this study. In Section 3.3 we describe related work by other open source researchers. In Section 3.4 we discuss our data source and data filtering methodology. In Section 3.5 we discuss interrater

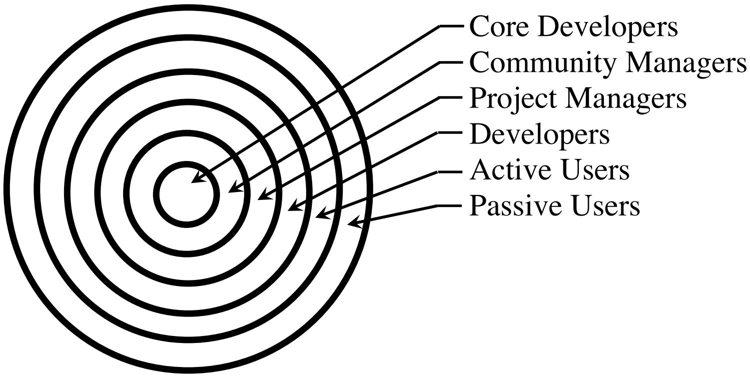


Figure 3.1: Onion diagram showing OSS project organizational hierarchy

reliability and interrater agreement. In Section 3.6 we discuss our tests for measuring reflexivity. In Section 3.7 we present our results, followed by threats to validity in Section 3.8 and general conclusions in in Section 3.9.

3.2 Previous Work

In a previous work [9], we presented preliminary findings on an analysis of SourceForge projects. We empirically tested Raymond's claim that "Every good work of software starts by scratching a developer's personal itch" [35, p. 25]. Empirically testing Raymond's statement required precise terminology. *Reflexivity*—a term we borrowed from sociology— is defined as "the capacity of language and of thought...to turn or bend back upon itself, to become an object to itself" [2, p. 2]. A reflexive action is one of self-reference, primarily intended to affect or benefit the doer. We adapted the term to the context of OSS as follows:

- **Reflexivity** – *The intent of developing software for the benefit of oneself or others like oneself* (i.e., for other developers).
- **Irreflexivity** – *The intent of developing software for the benefit of others not like oneself* (i.e., for passive users).

Reflexive software projects are primarily designed for developers, system administrators, or others in the field of information technology (rather than for casual users). Irreflexive projects are designed primarily for passive users, which may coincidentally include developers. Figure 3.2 shows the reflexive and irreflexive relationships between developers and users. Note that we focused on developer intent, rather than on end-user composition for two reasons: 1) Intent is easier to measure from available project artifacts; and 2) Developer intent drives participation in OSS projects.

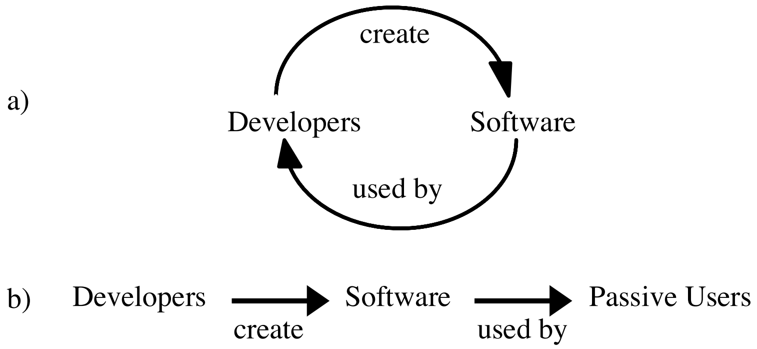


Figure 3.2: a) Reflexive and b) irreflexive software processes

For this paper, we extend our previous work by creating a statistical model through a random sampling of SourceForge projects. This improves on our previous study in three ways:

- We include more projects in our dataset (370,404 compared to 211,654).
- We include an analysis of how reflexivity has changed within SourceForge over time.
- We manually examine project information to classify projects as reflexive or irreflexive.
- We employ multiple reviewers to measure our rating consistency and accuracy.

Our previous work relied heavily on project metadata. Consequently, we could not classify projects that were missing metadata (for example, intended audiences, supporting operating systems, or relevant topics). Our automated approach found that 143,310 (68%) of the 211,654 SourceForge projects with metadata were reflexive.

Building on our previous work, we again reduce Raymond's statement ("Every good work of software starts by scratching a developer's personal itch") into testable hypotheses and add a third hypothesis to test whether the number of reflexive projects has increased over time. Our experimental hypotheses are outlined in Table 3.1.

Hypothesis	Informal Expression	Formal Expression
H1	Most projects are reflexive.	$P_R(DS) > 0.5$
H2	Reflexive projects are more successful than irreflexive projects.	$P_R(Down_j) > 0.5$
H3	Reflexivity has increased.	$P_R(Date_j) > 0.5$

P_R = Proportion Reflexive Projects, DS = Dataset,
$Down_j$ = Download Bucket j, $Date_j$ = Date Bucket j

Table 3.1: Hypotheses of Projects within SourceForge.

3.3 Related Work

Nichols and Twidale set up a framework to empirically study issues concerning OSS usability. In 2003, they examined the differences between developers and users [30]. In 2005, they studied bug reports of several OSS projects, exploring ways developers categorize users [31]. In 2006, they analyzed usability (and the lack of technical expertise on the part of users) in OSS projects [42].

Netta Iivari examined an end-user, media-player project on SourceForge [14]. Within this project, developers struggled to understand the final end-user of their software project. Moreover, their failure to properly identify their intended audience manifested itself in the terms they used when speaking of users. They spoke of *typical users*, *novice users*, *non-technical users*, and *users in general*—struggling to agree on which term properly identified their intended user. This confusion in terminology highlights a struggle in addressing user-related issues.

Singh et al. address the difficulties created by the so-called “curse of knowledge”—informed developers may struggle to understand the problems of those with less skill than themselves [40]. Accordingly, even if a project is developed and intended for use by less technically savvy users, the resulting lack of a well-designed interface can inhibit usability and subsequent appeal to its intended audience.

Rantalainen et al. claim that most open source projects are not designed for a general non-technical audience [34]. A typical user has neither the technical skill nor desire to use the vast majority of infrastructure and tools-based open source projects.

While the body of work identifying the lack of usability in OSS projects is growing, we find no large scale user-centered analysis of a large OSS repository, such as the one proposed here.

3.3.1 Success of Open Source Software

In general, success can be thought of as the degree to which a solution meets human goals or needs. Of course, directly quantifying human goals and needs is difficult. For commercial software, success is often measured by the profit generated from software sales. However, revenue models are less relevant in OSS contexts, since the source code is publicly released. Also such data would be difficult and time-consuming to obtain for such a large number of projects (even assuming that such financial data were available, which is not the case for most SourceForge projects). Consequently, we consider project metrics that are readily available in the SourceForge context—release frequency, developer activity, software quality, and downloads. Each of these metrics presents a different perspective on project success, but all are presumed to correlate at least minimally with meeting human goals and needs.

Success of OSS projects has been measured at various levels of analysis, including internal and external factors. Internal analysis examines factors relating to the nature of the OSS development process, such as growth in the size of the developer community [22], developer contribution [27], and activity level within open source projects [21] [32]. External analysis examines the products of OSS development, such as project popularity [44] and project usage/adoption (such as Netcraft's survey of Apache web server market share [29]), or—as will be used within our study—the number of times a project is downloaded.

Open source artifacts such as source code, mailing lists, and commit logs can be used in an internal analysis. Internal success can be measured by the motivation of developers [24] or ideology within the open source community [35]. Some research has focused on organizational structure concluding that an open source development process can produce high quality and widely distributed software [46]. Linux, Apache Server, and Eclipse are often cited as evidence of OSS development success.

3.3.2 SourceForge

SourceForge has been the subject of many previous studies. In 2002, Krishnamurthy analyzed SourceForge by counting the number of projects within six product development categories. He concluded that most mature projects were developed by a small number of developers [21]. In 2005, Rainer and Gale found few SourceForge projects actively being developed (less than 1%) [32]. In 2005, Weiss [45] showed ways that SourceForge organizes and represents data about projects. Building on these studies, English and Schweik classified SourceForge projects by release and developer activities [6].

3.4 Data Source

The University of Notre Dame hosts the SourceForge Research Data Archive[1] (SRDA), which provides researchers with a wide array of SourceForge metadata, pre-extracted into a usable database schema. We use the September 2011 SRDA data dump, which includes all SourceForge projects created from the inception of SourceForge through September 30, 2011. From this source we obtain all metadata considered in this paper, with the exception of downloads. The SRDA currently does not maintain up-to-date download statistics. For this information, we utilize SourceForge's API[2], from which we obtained download statistics for projects from November 4, 1999[3] through September 30, 2011 (i.e., covering the same

[1] http://zerlot.cse.nd.edu/mediawiki

[2] http://sourceforge.net/p/forge/documentation/Download%20Stats%20API/

[3] In the previous study we incorrectly used January 1, 2000 as our start date. While we did not miss any projects, we failed to gather 56,606 downloads from November 4, 1999 through December 31, 1999.

time range as the SRDA data). Our definition of *download* as a metric is consistent with the definition employed by the SourceForge API, namely any file downloaded from a project's file release space, independent of the contents or type of the file.

SourceForge data presents several challenges to our study. In May 2010 SourceForge began automatically generating a project entry each time a user account is created. Fortunately, these projects are designated with a 'u/' prefix on the project name and a '.u' suffix in the Unix group (SourceForge calls projects "groups") name—presumably 'u' means user. There are 145,928 projects in the SRDA database with the 'u/' prefix and 146,970 entries with '.u' suffix (the discrepancy of 42 between 'u/' and '.u' projects results from users changing their project name). These projects (which identify these as .u or profile projects) exist in the same table of data as other SourceForge projects, even though they more closely resemble a biographical webpage of the developer (i.e., a registered SourceForge user) than an open source project.

We excluded these user profile "projects" in our first study, concluding that they lacked the metadata necessary for our automated classification methodology. We again exclude 146,970 projects identified with a '.u' suffix in their Unix group name, based upon the following four factors:

1. The names of these projects appear to be the user's first and last name or the user's login id, suggesting they are auto-generated by SourceForge.
2. 146,092 (99.4%) projects have either a blank project description (126,383) or the default project description of "No description" (19,709).
3. 146,957 (99.999%) of "profile" projects have either a blank license (146,944) or "none" as their license (13).
4. 146,799 (99.9%) projects have either never been downloaded (146,220) or the project's download info is inaccessible through SourceForge's download API (579 projects).

In our previous study [9], we obtained download information only from projects that met our restricted metadata criteria. For this study, we increased our scope to obtain download counts for all projects that fall within our date range (Nov. 1999 through Sept 2011). Using SourceForge's API, we attempted to obtain the download counts for all projects in our dataset. However, we received a *404 Not Found* error for 3,494 projects. Of these projects, 579 are .u projects—leaving 2,915 projects unaccounted for.

To discover the nature of these 2,915 projects, we queried SourceForege's webserver to verify each project's development webpage. SourceForge organizes a project's development website (with the exception of the .u projects[4]) by their Unix group name according to the following URL pattern: `http://sourceforge.net/projects/[unix_group_name]`. However, we received a *404 Not Found* HTTP error for 2,914 of the 2,915 projects. It is possible that the development websites for these 2,914 projects are formatted differently than other SourceForge projects. To account for this, we used SourceForge's search feature and queried 50 randomly chosen projects within this "missing" project set—to no avail. Perhaps, as English and Schweik suggest, SourceForge "occasionally purges defunct projects" [6, p. 4]).

The only project that did not report download information but still has a working development website is the highly successful project `VLC`. SourceForge indicates millions of weekly downloads on the `VLC` development website, but we could discover no way to obtain an aggregated count for the number of times it has been downloaded. We include `VLC` in our dataset, but exclude it in answering Hypothesis H2, as this hypothesis is concerned with the number of times a project has been downloaded.

457,288 projects are accessible via the SRDA. We removed 149,884 projects (146,970 "user profile" projects and 2,914 "missing" projects) for a final dataset comprised of 307,404 projects.

[4] The development webpages of .u projects are formated as `http://sourceforge.net/u/[unix_group_name]`

3.5 Interrater Reliablity (IRR) and Interrater Agreement (IRA)

3.5.1 Background

An accurate measurement of reflexivity is subject to at least two threats: 1) The consistency of applying the definitions of reflexivity and irreflexivity to projects (i.e., precision) and 2) The consensus in the application of these definitions (i.e., accuracy).

Since we categorize a large number of projects, it is possible that we may rate one project as reflexive and another similar project as irreflexive; in other words, a certain amount of inconsistency is almost certain in the application of our definitions. To account for and deal with this concern, we employ the help of four external judges to review and categorize a subset of projects within our dataset. We then measure the agreement between ourselves and these judges to establish a statistic of consistency and reliability of categorizing projects as reflexive or irreflexive.

The correctness of categorizing a project as reflexive or irreflexive assumes a fixed reference point or "ground truth"—that is, a definitive measure of a project's reflexivity rating. However, no such absolute reference point exists. Instead, using the ratings of the external judges, we can measure how closely our ratings are interchangeable with the judges'. This provides a statistic for estimating our categorization correctness.

In 1960, Jacob Cohen created a statistic for measuring agreement between two judges rating along a nominal scale [4]. In 1971, Joseph Fleiss generalized Cohen's approach to cases with more than two judges [8]. Building on these approaches, James LeBreton and Jenell Senter analyzed various agreement measures and provided instructions for researchers to utilize these measures [23]. They cite two dominant agreement metrics: 1) Interrater agreement (IRA), which measures the "consensus in scores furnished by multiple judges for one or more targets" and 2) Interrater reliability (IRR), which measures the "relative consistency in ratings provided by multiple judges of multiple targets" [23, p. 816].

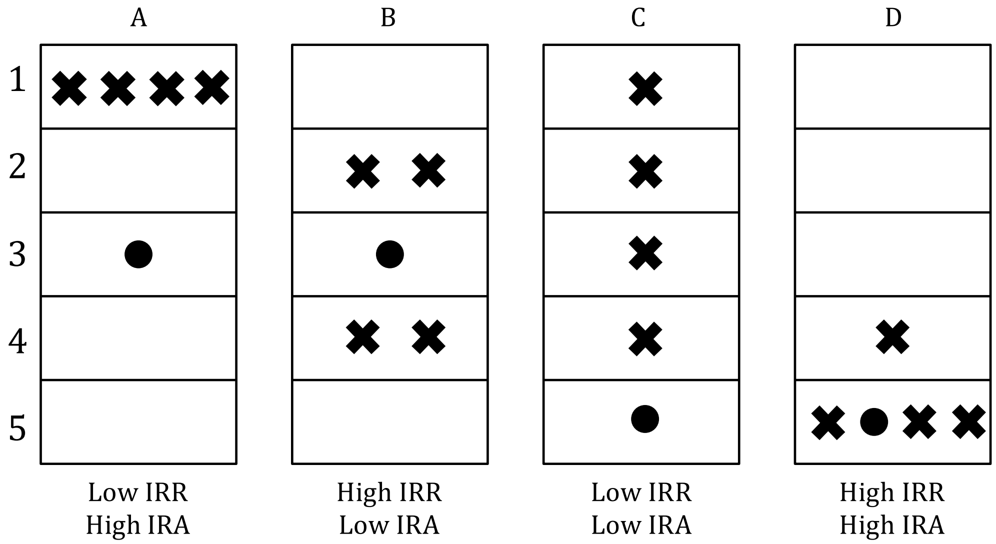


Figure 3.3: Four hypothetical examples of ratings by judges (shown as crosses) and the PI (shown as a circle) with corresponding hypothetical Interrater Reliability (IRR) and Interrater Agreement (IRA).

To differentiate between the ratings made by the lead author and those made by others, we will use the term *PI* to mean the principal investigator and the term *judges* to mean the four external judges.

The IRA metric measures how consistent the combined ratings of both the judges and the PI are at applying our definitions of reflexivity and irreflexivity. This provides a *qualitative* measure of the *qualitative* task of interpreting and applying our definitions of reflexivity and irreflexivity compared to the PI.

The IRR metric measures the similarity between the reflexivity scores of the PI and those of the judges, providing a qualitative measure of the PI's ability to categorize projects.

Figure 3.3 shows four possible hypothetical scenarios of ratings by the judges and the PI:

A Judges mostly agree (high IRA), but the PI's rating diverges from the mean of the judges' ratings (low IRR). This suggests that the judges match in their ability to apply the definition of reflexivity to a particular project, but the PI's interpretation of reflexivity does not match the judges'.

B Judges disagree (low IRA), but the PI's rating is close to the mean of the judges' ratings (high IRR). This occurs if the judges differ in their interpretation of reflexivity (as it relates to a particular project) but the PI's rating is similar to the mean of the judges' ratings.

C Judges cannot agree (low IRA) and the PI's rating is far from the mean of the judges' ratings (low IRR). This is the worst case scenario in which the judges all differ in applying the definition of reflexivity to a particular project and the PI's rating differs greatly from the mean of the judges' ratings.

D Judges mostly agree (high IRA) and the PI's rating is close to the mean of the judges' ratings (high IRR). This is the best case scenario in which the judges agree in their interpretation of reflexivity and the PI's rating is close to the mean of the judges' ratings.

3.5.2 Terms and Definitions

We now provide a brief overview of interrater agreement terms and assumptions in order to adequately define IRR and IRA.

We first consider three assumptions that Cohen stated are required to measure agreement (these have been recast in terms of our study): 1) projects are independent, 2) the nominal categories are independent, mutually exclusive, and exhaustive, and 3) the judges and the PI act independently [4, p. 38].

We comply with the first assumption (project independence) since SourceForge projects are independently created by developers.

To comply with the second assumption, we developed a nominal range to categorize projects, with a corresponding numerical code:

Code 0 - Not enough information

Code 1 - A reflexive project

Code 2 - Mostly a reflexive project

Code 3 - Half reflexive / half irreflexive

Code 4 - Mostly an irreflexive project

Code 5 - An irreflexive project

Judges were provided written instructions, directing them to codify a project as reflexive if it was "primarily designed for developers, system administrators, or others in the field of information technology." and codify a project as irreflexive if it was "primarily designed for passive users (i.e., users who are not interested in software development, but only in the resulting solution)."

The first iteration of our scale did not include Codes 2 and 4. However, after a trial run with another researcher (not one of the four external judges) we decided to include Codes 2 and 4. This researcher reported hesitancy in picking categories labeled "A reflexive project" and "An irreflexive project," based on a sense that to categorize a project as either reflexive or irreflexive would suggest that it must be *completely* reflexive or *completely* irreflexive. We decided a finer-grained scale would help judges in deciding whether a project is reflexive or irreflexive.

A common technique when dealing with lower-level nominal categories (i.e., finer-grained data) is to aggregate categories into composite higher-level categories (see [23] and [11] for a discussion on aggregating lower-level data). We do this because we are not concerned with the *degree* of reflexivity or irreflexivity, but simply whether a project is *more* reflexive than irreflexive or *more* irreflexive than reflexive. Aggregating Codes 1 and 2 and aggregating Codes 4 and 5 produces a better model with which to measure reflexivity and irreflexivity in SourceForge.

We aggregate Codes 1 and 2 as *Category Irreflexive*, Codes 4 and 5 as *Category Reflexive*, and Code 3 as *Category Half-Irreflexive / Half-Reflexive*[5]. Figure 3.4 shows the make-up of our three categories.

[5] We exclude projects tagged as Code 0 (*Not enough information*) since it is not a measure of whether a project is reflexive or irreflexive, but rather a reflection of a judge's inability to rate a project.

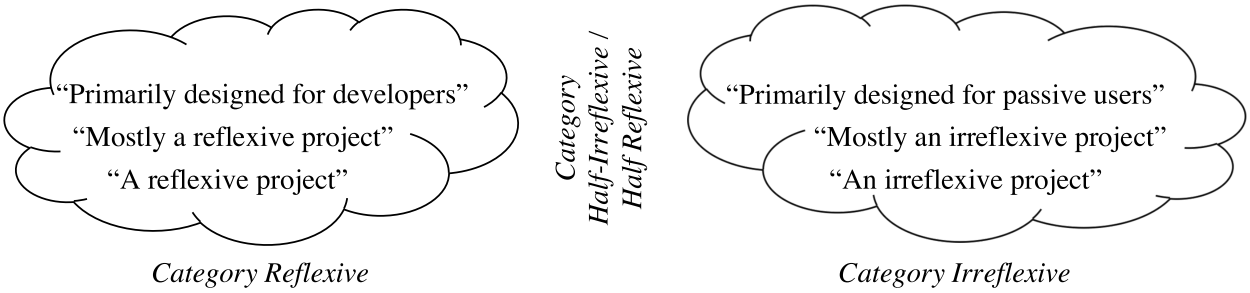

"Primarily designed for developers"
"Mostly a reflexive project"
"A reflexive project"
Category Reflexive
Category Half-Irreflexive / Half Reflexive
"Primarily designed for passive users"
"Mostly an irreflexive project"
"An irreflexive project"
Category Irreflexive

Figure 3.4: Make-up of Categories 1, 2, and 3

We took care to meet the third assumption (that the judges and PI act independently) by having the judges rate each project apart from any influence from the PI. Judges were provided both written instructions and written definitions of reflexivity and irreflexivity.

To measure IRR and IRA we considered two types of variances: *error variance* (also referred to as *observed variance*) and *expected variance*. Error variance is based on the assumption that each project has a *true* reflexivity score and any disagreement among judges is assumed to happen because of a judge's error in discerning the *true* score [17]. This variance we label as S^2 and is computed using the standard variance formula (i.e., the average of the squared differences from the mean). Perfect agreement results in $S^2 = 0$ and complete disagreement results in $S^2 = 1$.

Expected variance is based on the assumption that the differences between judges' ratings is based on random error (both Cohen and Fleiss based their variance calculations on this assumption). This variance we label as σ_E^2. Historically, a uniform null distribution has been used to determine the expected variance [23]. However, Lebreton and Senter note that:

> *...the uniform distribution is only applicable in situations where the scales are discrete and a complete lack of agreement would be hypothesized to be evenly distributed across response options (i.e., mathematically random)* [23, p. 829].

We did not expect judges to select Category Half-Reflexive / Half-Irreflexive with equal frequency as the other two categories. In fact, since judges were able to select along a range of reflexivity, we expected few projects to be categorized as half-reflexive and half-irreflexive.

Using a uniform distribution to calculate expected variance would over-inflate the probability of selecting Category Half-Reflexive / Half-Irreflexive. We expected Categories Irreflexive and Reflexive to be uniformly distributed using the assumption that expected variance is based on random chance. Therefore, we divided the random probability of choosing among the three categories as follows: Category Irreflexive = 0.475, Category Half-Reflexive / Half-Irreflexive[6] = 0.05, and Category Reflexive = 0.475. The expected variance is (using a mean of 2):

$$\sigma_E^2 = 0.475(1-2)^2 + 0.10(2-2)^2 + 0.475(3-2)^2 = 0.95. \tag{3.1}$$

If we had assumed a uniform[7] null distribution, the estimated variance would be 0.67, slightly lower than what we calculate in Equation 3.1.

To measure IRA, we use the multi-item index $r_{\mathrm{WG(J)}}$ [23, p. 818]. This is proportional to the difference between the variance due to error (S^2) and the variance due to random chance (σ^2), defined for individual projects as

$$r_{WG} = 1 - \frac{S^2}{\sigma^2}, \tag{3.2}$$

and overall as

$$r_{WG(J)} = \frac{J(1 - \frac{\bar{S_j^2}}{\sigma^2})}{J(1 - \frac{\bar{S_j^2}}{\sigma^2}) + (\frac{\bar{S_j^2}}{\sigma^2})}, \tag{3.3}$$

where J is equal to the number of projects categorized by judges and $\bar{S_j^2}$ is the mean of the observed variances for each jth project. Both of these equations produce an index value between 0 (total lack of agreement between judges) and 1 (total agreement between judges).

Gensheng Lui, et al., suggest in [25] that rather than using the mean of the observed variances of each jth project, as shown in Equation 3.3, the average of the r_{WG}s should be

[6] This assumption turned out to closely match our measured proportion. Of the 853 randomly chosen projects, 85 (4%) were categorized as Category Half-Reflexive / Half-Irreflexive.

[7] Uniform distribution is $(A^2 - 1)/12$, where A is equal to number of nominal categories.

used, which they label as R_{WG}:

$$R_{WG} = \frac{\sum_{j=1}^{N}(r_{WG_j})}{N}, \tag{3.4}$$

where N is equal to the number of projects categorized by the judges. They note that in checking the data of five studies that supposedly computed r_{WG}, the researchers actually computed R_{WG} instead. Rather than analyze the merits of which of these two approaches is best, we present both as a measure of IRA.

In 1990, George recommended using the value 0.7 for IRA indexes as the cutoff between high and low IRA [11] and this value has become *the* heuristic cutoff value of choice [23]. In 2011, Lui, et al. provide an analysis of the heuristic cutoff value and suggest that researchers conducting a study of a new relationship should use a higher cutoff value—presumably, since there is nothing to compare it against. We agree with LeBreton and Senter that a higher level than 0.7 provides more evidence that the "judges are truly seeing the same thing" [23].

A common method for measuring IRR is the Pearson product-moment correlation coefficient (commonly referred to as Pearson's r, see [23, p.822] and [38]), defined as

$$Pearson's\ r = \frac{\Sigma(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\Sigma(X_i - \bar{X})^2 \Sigma(Y_i - \bar{Y})^2}}, \tag{3.5}$$

where X_i is equal to the PI's rating for project i and Y_i is equal to a judge's rating for the same project. $\bar{X}$ is the mean of the ratings made by the PI and $\bar{Y}$ the mean of the ratings made by a judge.

Determining a "good" value for Pearson's r is highly subjective. To establish thresholds of "goodness" requires comparing the value to something else—perhaps against other similar experiments. Yet, as Cohen states, "The question, 'relative to what?' is not answerable concretely," since comparing one experiment's Pearson's r to another compares the *subjective* averaging and aggregating of one researcher's data to another [5, p. 79]. Given this understanding, we note that, similar to IRA, 0.7 has historically been used as the heuristic cutoff value [23].

We chose to limit the number of projects judges would rate to 100, first, because of the time commitment required to examine projects is significant, and second, because we conclude that ratings from 100 projects provide a large enough sample size for us to adequately measure IRA and IRR. We exclude a project if least one judge chose Code 0 (Insufficient information).

3.6 Measuring Reflexivity in SourceForge

We present three methods for measuring the number of reflexive and irreflexive projects within SourceForge[8]:

1. Entire Dataset (tests Hypothesis H1)
2. Download Buckets (tests Hypothesis H2)
3. Date Buckets (tests Hypothesis H3)

To test each of our hypotheses, we use a symmetrical-binomial test for goodness-of-fit (as described in [5, Ch. 5]). We compare the measured proportion of reflexive projects to a hypothesized proportion. Our hypothesis is that *most* projects are reflexive, therefore we set our hypothesized proportion to one-half (0.50)—our interpretation of "most." This hypothesis represents what we believe to be true and is characterized statistically as the *alternative hypothesis*. Consequently, the *null hypothesis* is that reflexive projects do not account for more than 50% of all projects.

We combine projects categorized as Half-Irreflexive / Half-Reflexive (half/half) with irreflexive projects. Such an approach ensures a conservative test by placing the half/half projects in such a way as to better ensure *failure to reject* the null hypotheses. In other words, if we reject our null hypotheses (that most projects are not reflexive), even when we

[8] We base much of our statistical analysis on the guidance of Jarrett Rosenberg found in [39, p.155-184] and Jacob Cohen found in [5].

place all half/half projects with irreflexive projects, we have stronger evidence to accept the alternative hypothesis (that most projects are reflexive).

Hyp.	Test	α	β	Power (1-β)	g (P-0.5)	Required N
$H1$	Binomial	0.05	0.10	0.90	0.05	853
$H2$	Binomial	0.05	0.10	0.90	0.10	210
$H3$	Binomial	0.05	0.10	0.90	0.10	210

Hyp. = Hypothesis, α = Type I Error significance,
β = Type II Error significance, g = Effective Size Index,
P = Measurable Proportion, N = Sample Size

Table 3.2: Statistical test values for each hypothesis

Table 3.2 shows the test values for each hypothesis. Required sample size is based on two parameters: power (how little Type II error) and effective size index (labeled as g). g is an index based on the sample size for detecting the true proportions, or in other words, the distance that the actual (true) proportion of reflexive projects is from our hypothesized proportion (0.50). For example, to detect if the true proportion of reflexive projects is greater than 0.55, g would be 0.05 (0.55 - 0.50). Table 5.4.1 in [5, p. 167] contains the required sample size for various values of g.

Below, we describe the three methods and our computation of g and sample size for each method.

3.6.1 Measuring Reflexivity in Dataset

To test Hypothesis H1, we set our statistical significance levels α to 0.05 (Type I error) and β (Type II error) to 0.10; our power is then 0.9. We wish to detect if the true proportion of reflexive projects is within 5% of 0.50. Therefore, we set g to 0.05.

We randomly selected 853 projects[9] from the entire dataset and categorized each project according to our nominal scale. The results of our binomial test is presented in Section 3.7.2.

3.6.2 Measuring Success

At least two types of downloads can be examined: source code downloads and project binary downloads. Developers download source code to work on a project. Users download a binary that the developers release. Creating a binary file available for users to download demonstrates a minimal level of success. And measuring the number of downloads provides insight into user demand. Despite its limitations (which we discuss in Section 3.8), we believe downloads is, in fact, a reasonable metric—in particular, because downloads directly reflects user demand. Further, if Raymond is right, one would expect project reflexivity to positively impact downloads.

Given that 502 projects account for 82% of all downloads [9], we employ a block sampling approach in which we group projects into buckets based upon download counts. This sampling approach ensures representation from both frequently and infrequently downloaded projects. Since the distribution of downloads is normally distributed along a log_{10} scale (excepting Download Bucket ND—projects that have never been downloaded), we place projects into buckets using powers of ten. Table 3.3 shows each bucket with its corresponding range. We also show the number of projects from the dataset which fall into each bucket.

To test Hypothesis H2, we set our statistical significance levels α to 0.05 (Type I error) and β (Type II error) to 0.10; our power is then 0.9. We set g to 0.10—a slightly higher value than used in our entire dataset test (H1). With eleven Download Buckets, setting g equal to 0.05 (g for our entire dataset test) would entail categorizing over 9,383 projects.

[9] We knew from our previous study that SourceForge contains projects with no project descriptions or descriptions that lacked sufficient details to determine whether the project was reflexive or irreflexive. To account for this, we initially randomly selected more than the required number of projects. We tagged these projects as *insufficient information* and removed them from the pool of projects. We then continued our categorization process until we had reached the required number of projects.

Download Bucket	Range	Projects in Bucket
ND	$D = 0$	175,866
0	$0 < D <= 10^0$	1,517
1	$10^0 < D <= 10^1$	6,508
2	$10^1 < D <= 10^2$	21,823
3	$10^2 < D <= 10^3$	54,210
4	$10^3 < D <= 10^4$	33,591
5	$10^4 < D <= 10^5$	10,945
6	$10^5 < D <= 10^6$	2,435
7	$10^6 < D <= 10^7$	423
8	$10^7 < D <= 10^8$	76
9	$10^8 < D$	9
	Total	307,403

D = downloads, ND = no downloads

Table 3.3: Download Buckets (Project `VLC` is excluded)

Though we accept a slightly higher g index value, 0.10 is still considered a small value by conventional definitions (see [5, p. 151]).

With a g index of 0.10 and power of 0.9, the required sample size is 210. We randomly selected 210 projects[10] (with two exceptions: Download Bucket 8 has only 76 projects and Download Bucket 9 which only has 9 projects) and categorized each project according to our nominal scale. The results of our binomial test is presented in Section 3.7.3.

3.6.3 Measuring Change in Reflexivity Over Time

To test whether the percentage of reflexive projects has increased over time (Hypothesis H3), we group projects into one year buckets based upon their registration date. Our date buckets begin November 1999 (the first SourceForge project registration). Because the first SourceForge project was registered in November, we chose our year range to go from November to the November of the following year (the exception is that the last project in our dataset

[10] See footnote 9.

was created on September 30th). Table 3.4 shows the date range of each Date Bucket as well as the total number of projects in our dataset that fall within the stated range.

Date Bucket	Range	Projects in Bucket
2000	Nov 04, 1999 $> RD <=$ Nov 30, 2000	5,273
2001	Nov 30, 2000 $> RD <=$ Nov 30, 2001	9,397
2002	Nov 30, 2001 $> RD <=$ Nov 30, 2002	13,243
2003	Nov 30, 2002 $> RD <=$ Nov 30, 2003	15,157
2004	Nov 30, 2003 $> RD <=$ Nov 30, 2004	18,419
2005	Nov 30, 2004 $> RD <=$ Nov 30, 2005	20,760
2006	Nov 30, 2005 $> RD <=$ Nov 30, 2006	27,914
2007	Nov 30, 2006 $> RD <=$ Nov 30, 2007	26,816
2008	Nov 30, 2007 $> RD <=$ Nov 30, 2008	27,769
2009	Nov 30, 2008 $> RD <=$ Nov 30, 2009	36,707
2010	Nov 30, 2009 $> RD <=$ Nov 30, 2010	57,523
2011	Nov 30, 2010 $> RD <=$ Sept 30, 2011	48,426
	Total	307,404

RD = registration date

Table 3.4: Date Buckets

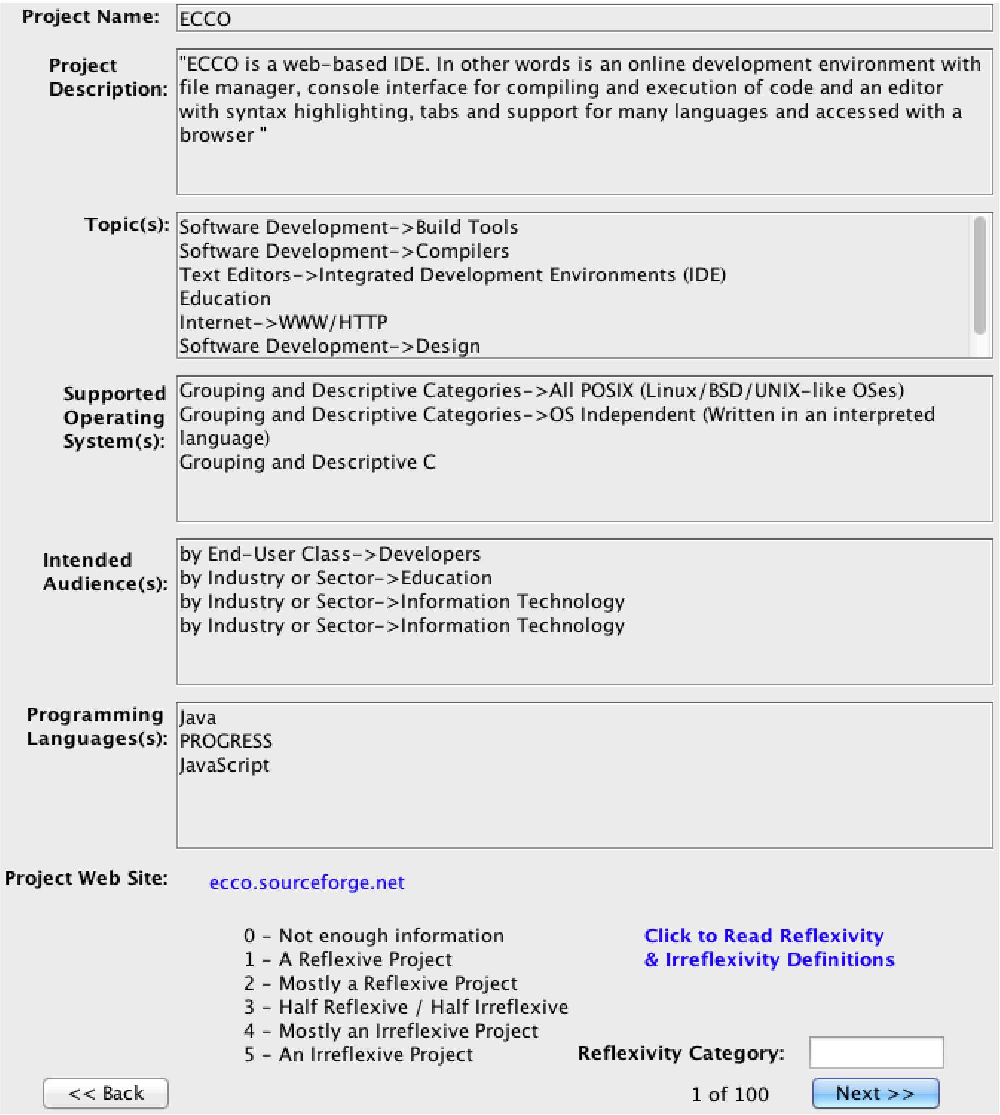
To test Hypothesis H3, we set our statistical significance levels α to 0.05 (Type I error) and β (Type II error) to 0.10; our power is then 0.9. We wish to detect if the true proportion reflexive projects is within 10% of 0.50. Therefore, we set g to 0.10. Our choice of g is based on the same reasoning as explained in the above section for Download Buckets.

With a g index of 0.10 and power of 0.9, the required sample size is 210. We randomly selected 210 projects[11] and categorized each project according to our nominal scale. The results of our binomial test is presented in Section 3.7.4.

3.6.4 Reflexivity Categorization Tool

We developed an interface tool to aid judges in categorizing projects. Figure 3.5 shows an image of our tool. The categorization tool presents a project's name, description, topic,

[11] See footnote 9.



Project Name: ECCO

Project Description: "ECCO is a web-based IDE. In other words is an online development environment with file manager, console interface for compiling and execution of code and an editor with syntax highlighting, tabs and support for many languages and accessed with a browser "

Topic(s):
Software Development->Build Tools
Software Development->Compilers
Text Editors->Integrated Development Environments (IDE)
Education
Internet->WWW/HTTP
Software Development->Design

Supported Operating System(s):
Grouping and Descriptive Categories->All POSIX (Linux/BSD/UNIX-like OSes)
Grouping and Descriptive Categories->OS Independent (Written in an interpreted language)
Grouping and Descriptive C

Intended Audience(s):
by End-User Class->Developers
by Industry or Sector->Education
by Industry or Sector->Information Technology
by Industry or Sector->Information Technology

Programming Languages(s):
Java
PROGRESS
JavaScript

Project Web Site: ecco.sourceforge.net

0 - Not enough information
1 - A Reflexive Project
2 - Mostly a Reflexive Project
3 - Half Reflexive / Half Irreflexive
4 - Mostly an Irreflexive Project
5 - An Irreflexive Project

Reflexivity Category:

1 of 100

Figure 3.5: A screen shot of the tool judges used to categorize projects. The tool shows a project's name and description, along with developer selected topics, supported operating systems, intended audiences, and programming languages. A clickable link opens the project development website.

operating system, intended audience, programming language, and a clickable link to each project development website. Judges were tasked with reading the project information and selecting one of the six categories. The definitions of reflexivity and irreflexivity were available to the judges for reference.

Using the categorization tool, the PI examined and categorized 5,348 randomly selected projects: 853 from the entire dataset, 1,975 from the Download Buckets, and 2,520 from the Date Buckets.

The tool was sent to the four judges (three computer science undergraduate students and one undergraduate electrical engineering student) with a random sample of 100 projects, and their responses collected.

3.7 Results

In Section 3.7.1 we address interrater agreement by measuring IRR and IRA using the project ratings of the PI and the judges. In Section 3.7.2 we examine reflexivity in the entire SourceForge population by randomly sampling projects. We address our first hypothesis (H1), regarding whether OSS projects tend to be reflexive (as Raymond insinuated) or irreflexive. In Section 3.7.3, we examine irreflexivity and reflexivity within each Download Bucket. We address our second hypothesis (H2) regarding whether irreflexive OSS projects tend to be more successful than reflexive projects. In Section 3.7.4 we examine reflexivity within each Date Bucket. We address our third hypothesis (H3), regarding whether the percentage of irreflexive projects has increased over time.

3.7.1 Interrater Reliability (IRR) and Interrater Agreement (IRA)

Figure 3.6 shows both project ratings of the PI and the judges for the 95 randomly selected projects (five projects were removed based on one or more judges selecting Code 0—*Insufficient information*), as well as the results of r_{WG} (Equation 3.2). Projects are shown along the horizontal-axis and the left vertical-axis lists the three nominal categories. The right vertical-axis is indexed from 0 to 1 to show the computed index value r_{WG} of each project. The mean and median values of the four judges is charted together with the PI's selected project rating to give a sense of how closely the PI and the judges agree. We ordered the results according to the PI's selected project rating, from Category Reflexive to Category Irreflexive. After

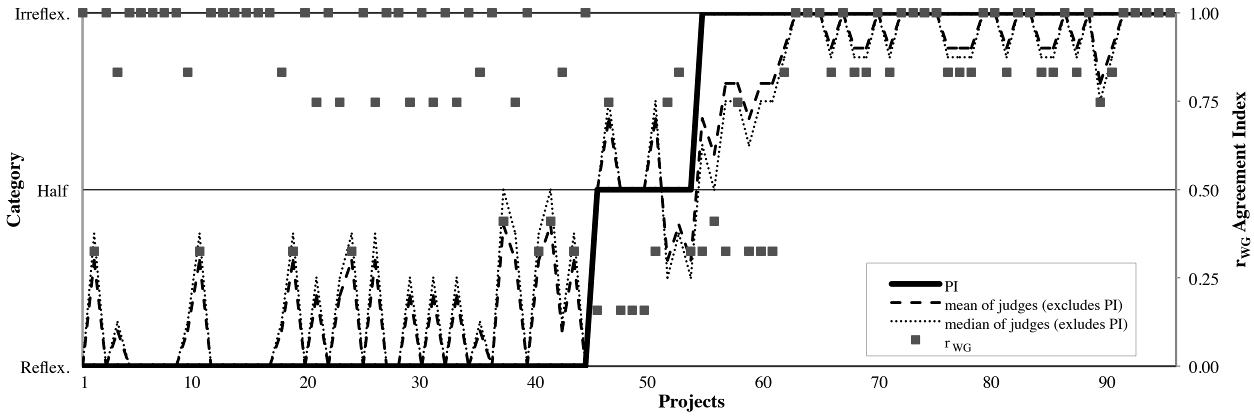


Figure 3.6: Interrater Agreement (IRA). Reflex. = Reflexive, Half = Half-Reflexive / Half-Irreflexive, Irreflex. = Irreflexive. The PI rating for each project is shown compared to the mean and median ratings of the judges. Index r_{WG} is also shown for each project. Projects have been ordered by the PI's rating in ascending order for clarity in comparing to the judges' ratings.

ordering, each project was assigned a project number from 1 to 95 to aid in comparing and graphing the results.

As computed in Equation 3.1, σ_E^2 is 0.95. The overall agreement index $r_{WG(J)}$ (Equation 3.3) is 0.997 and R_{WG} (Equation 3.4) is 0.789. r_{WG} is significantly higher since it uses the mean of the observed variance. Index R_{WG} uses the observed variance of the judges' ratings for each project and then averages the results, producing a lower value. In either case, the overall index value is above 0.7, which we interpret as a strong to very strong agreement among the judges (high IRA).

Twenty projects have a r_{WG} index value of less than 0.70. There were twelve occurrences of one rater (raters include the four judges and the PI) rating a project as Category Irreflexive or Reflexive when the other raters rated the project as respectively Category Reflexive or Irreflexive. The other eight occurrences result from two judges likewise rating differently from the consensus of the other raters.

We can only speculate as to why some judges categorized a project as reflexive when the majority categorized the same project as irreflexive (or mixed). One possibility is that judges differed in their interpretation of highly technical projects. For instance, one project in which judges differed is a security solution for accessing public library computers running Linux. The PI and two other judges rated this as Category Half-Irreflexive / Half-Reflexive, while the other two judges rated it as Category Irreflexive. It is unclear from the project information whether this project is intended for use by librarians or system administrators. If intended for librarians then perhaps it should be rated Category Irreflexive or Category Half-Irreflexive / Half-Reflexive; if intended for system administrators, then it should be rated Category Reflexive.

	PI	J1	J2	J3	J4		
Cat. Reflex.	44	45	46	35	32		
Cat. Half	9	16	16	10	17	J_i	
Cat. Irreflex.	44	33	32	49	45	Mean	Median
Pearson's r	-	0.790	0.779	0.833	0.806	0.924	0.940

Cat. = Category, Reflex. = Reflexive, Half = Half-Irreflexive / Half-Reflexive, Irreflex. = Irreflexive

Table 3.5: Rating counts for PI and each judge per category along with the Pearson's r coefficient.

Overall, there is high agreement between the mean (and median) of the judges and the PI as computed by Pearson's r, as shown in Table 3.5. The Pearson's r coefficients for Judges 1, 2, 3, and 4 are above 0.70, suggesting strong correlation and consistency between each Judge and the PI. The mean and median of the judges' project ratings are shown in the last two columns. The mean and median values very nearly match the PI's ratings and produce a very high Pearson's r coefficient of, respectively, 0.924 and 0.940 (1.0 being perfect agreement).

Another strong indicator of similarity between the ratings of the PI and the judges, as can be seen on Figure 3.6, is that computing the judge's mean (or median) never resulted in a project being categorized as reflexive when the PI categorized it as irreflexive; similarly, there was never a time when computing the judge's mean (or median) resulted in a project being categorized as irreflexive when the PI called it reflexive. Given that the judges were unfamiliar with our definitions of reflexivity and irreflexivity when they started, we are not surprised to find discrepancies between judges and the PI. It seems probable that the collective wisdom of the judges (i.e., their mean and median values) should approximate the expertise of the PI—though this is of course conjecture on our part.

An examination of individual judges shows some interesting tendencies. Both Judges 1 and 2 rated five more projects as Category Half-Irreflexive / Half-Reflexive when compared to the PI, but rated nearly the same number of projects as a Category Reflexive. From this we conclude that Judges 1 and 2 were less willing to categorize projects as irreflexive when compared to the PI. Similarly, Judges 3 and 4 rated, respectively, one and eight more projects as Category Half-Irreflexive / Half-Reflexive than the PI but rated nearly the same number as Category Irreflexive. From this we conclude that Judges 3 and 4 were less willing to categorize projects as reflexive when compared to the PI.

We interpret these results as strong evidence of *consensus* (i.e., IRA) between raters in applying our definitions of reflexivity and irreflexivity and strong evidence of *correlation* (i.e., IRR) between the PI's ability to categorize projects and the collective wisdom of the judges. We conclude that these results are similar to example D in Figure 3.3 which graphically depicts both high IRA and high IRR.

3.7.2 Hypothesis H1: Reflexivity in SourceForge (Entire Dataset)

Table 3.6 contains the results of our statistical tests[12]. The table is divided by the three bucket types we are testing: Entire Dataset, Download Buckets, and Date Buckets. The number of projects within each bucket is provided along with the number and proportion of projects that are reflexive, irreflexive & half/half. The last three columns contain our statistical test values for the lower and upper 95% confidence intervals, and p-values for each bucket. The p-value refers to the null hypothesis of whether the proportion of reflexive projects is less than or equal to 0.50. A p-value of less than 0.05 is evidence to reject the null

[12] We used statistical software package JMP version 10.0.0 to compute all our statistical findings.

	Bucket	N	Reflex.	Irreflex.	Half / Half	Prop. Reflex.	Prop. Irreflex. & Half/Half	CI-L	CI-U	p-Value$^{\Psi}$
	Dataset	853	475	346	32	0.56	0.44	0.52	0.59	**(0.001)**
Download Bucket	ND	210	112	76	22	0.53	0.47	0.47	0.60	0.185
	0	210	117	69	24	0.56	0.44	0.49	0.62	0.056
	1	210	118	72	20	0.56	0.44	0.49	0.63	**0.042**
	2	210	138	62	10	0.66	0.34	0.59	0.72	**(0.001)**
	3	210	122	80	8	0.58	0.42	0.51	0.65	**0.011**
	4	210	124	73	13	0.59	0.41	0.52	0.65	**0.005**
	5	210	132	69	9	0.63	0.37	0.56	0.69	**(0.001)**
	6	210	114	86	10	0.54	0.46	0.48	0.61	0.120
	7	210	118	85	7	0.56	0.44	0.49	0.63	**0.042**
	8*	76	43	31	2	0.57	0.43	-	-	-
	9*	9	7	2	0	0.78	0.22	-	-	-
Date Bucket	2000	210	132	63	15	0.63	0.37	0.56	0.69	**(0.001)**
	2001	210	146	51	13	0.70	0.30	0.63	0.75	**(0.001)**
	2002	210	126	65	19	0.60	0.40	0.53	0.66	**0.002**
	2003	210	141	63	19	0.67	0.33	0.61	0.73	**(0.001)**
	2004	210	119	77	14	0.57	0.43	0.50	0.63	**0.031**
	2005	210	106	96	8	0.50	0.50	0.44	0.57	0.473
	2006	210	114	89	7	0.54	0.46	0.48	0.61	0.120
	2007	210	129	71	10	0.61	0.39	0.55	0.68	**0.001**
	2008	210	120	77	13	0.57	0.43	0.50	0.64	**0.023**
	2009	210	106	83	21	0.50	0.50	0.44	0.57	0.473
	2010	210	107	76	27	0.51	0.49	0.44	0.58	0.418
	2011	210	103	66	41	0.49	0.51	0.42	0.56	0.635

N = Sample size, Irreflex. = Irreflexive, Reflex. = Reflexive, Prop. = Proportion, (0.001) = less than 0.001, CI-L = Lower 95% Confidence Interval, CI-U = Upper 95% Confidence Interval, $^{\Psi}H_O$: Proportion Reflexive Projects ≤ 0.5
*Complete populations (all projects) of Download Buckets 8 and 9 were categorized (see Section 3.7.3).

Table 3.6: Summary results of randomly sampled projects in SourceForge

hypothesis and accept the alternative hypothesis that the proportion of reflexive projects is greater than 0.50.

The row labeled "Dataset" shows the number and proportion of reflexive projects and irreflexive & half/half projects. Based on these proportions and on our statistical test values (α and β), we confirm Hypothesis $H1$ by rejecting the null hypothesis. Thus, we accept the alternative hypothesis that reflexive projects make up more than 50% of all projects—in this case, 56% of SourceForge projects are reflexive.

While the difference between the proportion of reflexive and irreflexive projects is statistically significant there is not a large practical difference.

3.7.3 Hypothesis H2: Reflexivity in Download Buckets

Each Download Bucket is shown in Table 3.6 with the number and proportion of reflexive projects and irreflexive & half/half projects. Based on these proportions, we confirm Hypothesis $H2$ for buckets 1 through 5 and 7; that is, reflexive projects make up more than 50% of projects in these buckets. Since we were able to categorize every project in buckets 8 and 9, we do not need to perform a goodness-of-fit test. Based on the actual proportions within buckets 8 and 9, we conclude that there are more reflexive projects in both buckets.

Hypothesis $H2$ does not hold for buckets ND, 0 and 6; that is, reflexive projects do not make up more than 50% of projects within these buckets. Since buckets 0 and 6 have p-values close to 0.05, if a few of the half/half projects were moved into the reflexive column then we would obtain a p-value of less than 0.05. Thus, if we were slightly less conservative in our approach, we may reach the conclusion that buckets 0 and 6 are mostly reflexive.

Though we confirm Hypothesis $H2$ for many of the Download Buckets, we do not see the proportion of reflexive projects increasing from one bucket to the next. In other words, we do not find a correlation between the number of downloads and reflexivity.

3.7.4 Hypothesis H3: Reflexivity in Date Buckets

Each Date Bucket is shown in Table 3.6 with the number and proportion of reflexive projects and irreflexive & half/half projects. Based on these proportions, we confirm Hypothesis $H3$ for Date Buckets 2000 through 2004, 2007, and 2008; that is, reflexive projects make up more than 50% of projects in these buckets.

Hypothesis $H2$ does not hold for buckets 2005, 2006, and 2009 through 2011. There are more irreflexive projects in bucket 2005 and 2006 compared to other years, and more half/half projects in 2009 through 2011 compared to other years. This suggests more irreflexivity in later years.

3.8 Threats To Validity

In our first study we identified three threats to the validity of our study. We describe each of these threats and how we dealt with them in this new study.

First, we examined fewer projects in our original study 211,654 projects compared to 370,404 in this study. One reason for the smaller dataset was that we excluded projects with missing metadata necessary to categorize projects as reflexive. We recommended changing the methodology to include projects with missing metadata. In this study, we did not exclude projects with missing metadata. We found many projects with missing project descriptions or projects with descriptions that lacked sufficient details for us to categorize as reflexive or irreflexive. In the original study, we also excluded .u projects (as described in Section 3.4). We investigated further and concluded these are auto-generated profile pages created when users register an account on SourceForge.

Second, we identified cases of project binaries that are bundled and distributed with other software (e.g., some open source utilities are distributed with certain versions of Linux). Our download metric failed to capture all instances where a project is used, hence potentially under-measuring project success. Dealing with this threat would require determining which projects are bundled elsewhere. Given the time and resources available to us, such an undertaking is outside the scope of our study. Another shortcoming was in using the term "download," which insinuates that users are intentionally downloading an OSS project, which may not be true. Again, obtaining the intentions of users is outside the scope of this study.

Using downloads as a raw count ignores the fact that some very successful projects may apply only to a small niche, with relatively high success reflected in a low number of downloads. We accounted for this by placing projects into buckets and measuring the proportion of reflexive and irreflexive projects within each bucket. This permits correlating success with reflexivity by comparing projects with similar download counts.

Third, our classification of SourceForge projects is a subjective judgment and we recommended refining our classification methodology through the use of multiple reviewers. We identified a potential for bias in finding reflexivity because we were looking for it, a kind of *self-fulfilling prophecy*. To account for this we employed the help of external judges to rate projects and compare their ratings to ours. We find consistency between the judges and correlation between the ratings of the judges and our ratings.

Lastly, in our first study we used *inclusion* criteria to classify projects with respect to reflexivity. We noted that searching for only positive cases may inflate the results. We recommended using both inclusionary and exclusionary criteria. To account for this, we developed a nominal rating scale that included definitions of both irreflexivity and reflexivity. Raters were allowed to rate projects as reflexive, irreflexive or both. This allowed us to test for both irreflexivity and reflexivity.

3.9 Conclusions

Open source users have traditionally been thought of as similar to (or even synonymous with) developers. From Hypothesis H1 we conclude that the majority of open source SourceForge projects are intended for a developer audience. Yet, from Hypothesis H3, we find a slight shift in the intended audience from reflexive (developer facing) to irreflexive (passive user facing), reflecting a cultural change from the dawn of open source software. Raymond's statement regarding developers scratching their own itch may have been true when Raymond made it, but we find, over time, more projects targeting non-developers.

We do not find a correlation between success (number of downloads) and reflexivity as we found in our first study. Reflexive projects do make up more than 50% of projects within most Download Buckets, but this is explained by the fact that the majority of SourceForge projects are reflexive. However, it may be that changes over time in the prevalence of reflexivity are masking the correlation between success and reflexivity. Thus future work requires analyzing this relationship as a function of time.

References

[1] R. Arif and L. Capretz. Do open source software developers listen to their users? *First Monday*, 17(3-5), March 2012.

[2] B. A. Babcock. Reflexivity: Definitions and discriminations. *Semiotica*, 30(1-2):1–14, 1980.

[3] A. M. Braccini, C. Silvestri, and S. Za. Interactions with open source software: A pilot study on end users' perception. In Alessandro D'Atri and Domenico Saccà, editors, *Information Systems: People, Organizations, Institutions, and Technologies*, pages 549–556. Physica-Verlag HD, 2010.

[4] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[5] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Academic Press, Inc., 1977.

[6] R. English and C. M. Schweik. Identifying success and tragedy of FLOSS commons: A preliminary classification of Sourceforge.net projects. In *Proc. Int'l Workshop Emerging Trends in FLOSS Research and Development*, 2007.

[7] J. Feller and B. Fitzgerald. A framework analysis of the open source software development paradigm. In *Proc. Int'l Conf. Information Systems*, ICIS '00, pages 58–69, Atlanta, GA, 2000. Association for Information Systems.

[8] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[9] B. Foushee, J. L. Krein, J. Wu, R. Buck, C. D. Knutson, L. J. Pratt, and A. C. MacLean. Reflexivity, raymond, and the success of open source software development: A sourceforge empirical study. In *Proc. Int'l Conf. on Evaluation and Assessment in Softw. Eng.*, April 2013.

[10] M. D. Gallego, P. Luna, and S. Bueno. User acceptance model of open source software. *Computers in Human Behavior*, 24(5):2199–2216, 2008.

[11] J. M. George. Personality, affect, and behavior in groups. *Journal of Applied Psychology*, 75(2):107–116, 1990.

[12] K. Healy and A. Schussman. The ecology of open-source software development. Technical report, University of Arizona, Tucson, AZ, 2003.

[13] E. Hippel. *Democratizing Innovation*. The MIT Press, 2005.

[14] N. Iivari. "Constructing the users" in open source software development: An interpretive case study of user participation. *Information Technology & People*, 22(2):132–156, 2009.

[15] N. Iivari. Empowering the users? A critical textual analysis of the role of users in open source software development. *AI & Society*, 23:511–528, 2009.

[16] N. Iivari, H. Hedberg, and T. Kirves. Usability in company open source software context: Initial findings from an empirical case study. In *Open Source Development, Communities and Quality*, volume 275 of *IFIP Int'l Federation for Information Processing*, pages 359–365. Springer, 2008.

[17] L. R. James, R. G. Demaree, and G. Wolf. rwg: An assessment of within-group interrater agreement. *Journal of Applied Psychology*, 78(2):306, 1993.

[18] C. Jensen and W. Scacchi. Role migration and advancement processes in OSSD projects: A comparative case study. In *Softw. Eng., ICSE 2007. 29th Int. Conf.*, pages 364–374, May 2007.

[19] L. R. Karim and E. Hippel. How open source software works: "free" user-to-user assistance. *Research Policy*, 32(6):923–943, 2003.

[20] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[21] S. Krishnamurthy. Cave or community? An empirical examination of 100 mature open source projects. *First Monday*, 7(6), June 2002.

[22] G. Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, 2003.

[23] J. M. LeBreton and J. L. Senter. Answers to 20 questions about interrater reliability and interrater agreement. *Organizational Research Methods*, 11(4):815–852, 2008.

[24] J. Lerner and J. Tirole. Some simple economics of open source. *The Journal of Industrial Economics*, 50(2):197–234, 2002.

[25] G. J. Liu, M. M. Amini, E. Babakus, and M. B. R. Stafford. Handling observations with low interrater agreement values. *Journal of Analytical Sciences, Methods and Instrumentation*, 1(2):9–18, 2011.

[26] G. Madey, V. Freeh, and R. Tynan. The open source software development phenomenon: An analysis based on social network theory. In *Americas Conf. on Information Systems (AMCIS2002)*, pages 1806–1813, 2002.

[27] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11:309–346, July 2002.

[28] A. Mockus, R.T. Fielding, and J. Herbsleb. A case study of open source software development: The Apache Server. In *Softw. Eng. Proc. 2000 Int'l Conf.*, pages 263–272, 2000.

[29] Netcraft. `http://news.netcraft.com/archives/2006/04/26/apache_now_the_leader_in_ssl_servers.html`. [Online; accessed 16-March-2013], 2006.

[30] D. Nichols and M. Twidale. The usability of open source software. *First Monday*, 8(1-6), Jan 2003.

[31] D. Nichols and M. Twidale. Usability processes in open source projects. *Software Process: Improvement and Practice*, 11(2):149–162, 2006.

[32] A. Rainer and S. Gale. Evaluating the quality and quantity of data on open source software projects. In *Proc. Int'l Conf. Open Source Systems*, pages 29–36, 2005.

[33] F. L. Ramsey and D. W. Schafer. *The statistical sleuth: a course in methods of data analysis*. Brooks/Cole, Cengage Learning, second edition, 2002.

[34] A. Rantalainen, H. Hedberg, and N. Iivari. A review of tool support for user-related communication in FLOSS development. In *Open Source Systems: Grounding Research*, volume 365 of *IFIP Advances in Information and Communication Technology*, pages 90–105. Springer Boston, 2011.

[35] E. Raymond. The cathedral and the bazaar. *Knowledge, Technology, & Policy*, 12(3):23–49, 1999.

[36] E. Raymond. The revenge of the hackers. *Open Sources–Voices from the Open Source Revolution', O 'Reilly, Cambridge, MA, USA*, pages 207–220, 1999.

[37] A. Raza, L. Capretz, and F. Ahmed. Users' perception of open source usability: an empirical study. *Eng. with Computers*, 28:109–121, 2012.

[38] J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[39] F. Shull, J. Singer, and D. I. K. Sjøberg. *Guide to Advanced Empirical Softw. Eng.* Springer-Verlag London Limited, 2008.

[40] V. Singh, M.B. Twidale, and D.M. Nichols. Users of open source software – how do they get help? In *System Sciences, 2009. HICSS '09. 42nd Hawaii Int'l Conf.*, pages 1–10, Jan 2009.

[41] L. Torvalds. The torvalds transcript: Why I 'absolutely love' GPL version 2, March 2007.

[42] M. B. Twidale and D. M. Nichols. Exploring usability discussion in open source development. In *Proc. 38th Hawaii Int'l Conf. System Sciences, HICSS-38, Track 7. 198c*, page 198, 2005.

[43] V. Valloppillil. Open source software: A (new?) development methodology. Online; accessed 07-March-2013 (also referred to as The Halloween Document), October 1998.

[44] D. Weiss. Measuring success of open source projects using web search engines. *Proc. 1st Int'l Conf. Open Source Systems*, July 2005.

[45] D. Weiss. Quantitative analysis of open source projects on SourceForge. In *Proc. Int'l Conf. Open Source Systems*, pages 140–147, 2005.

[46] Y. Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *Proc. Int'l Conf. Softw. Eng.*, pages 419–429, 2003.