Quantitative Study of Open Source Immigration

Christian Bird, Alex Gourley, Prem Devanbu
Dept. of Computer Science
UC Davis
Davis, CA 95616,USA
cabird,acgourley,devanbu@ucdavis.edu

Anand Swaminathan, Greta Hsu
Graduate School of Business
UC Davis
Davis, CA 95616, USA
aswaminathan,grhsu@ucdavis.edu

Abstract

Open source software is built by teams of volunteers. Each project has a core team of developers who have the authority to commit changes to the repository; this team is the elite core of the project, selected through a meritocratic process from a larger number of people who participate on the mailing list. Understanding the factors that influence the "who, how and when" of this process is critical for the sustainability of FLOSS projects and for outside stakeholders who want to gain entry and succeed. Prior research indicates that certain types of behaviors, such as the duration and intensity of participation on the developer mailing list, and the submission of patches, play a role in immigration. However, this work has largely been qualitative and/or descriptive. We use quantitative hazard rate modeling, which supports the statistical testing of hypotheses concerning the influence of these factors on the rate of immigration. We develop a theory of open source project joining, and use data from Postgres, Python, and the Apache web server to statistically evaluate it using a piecewise-constant proportional hazard rate model. Quantitative modeling reveals variations across the projects in the effects of a participant's a) duration in a FLOSS community; b) their volume of patch submission; and c) and their social status. These variations can be attributed to differences across the projects in institutional norms for joining, technical complexity of the projects and governance mechanisms.[1]

1. Introduction

Brooks [6], in his classic *The Mythical Man-Month* made an enduring observation about the risks of adding new people into software project teams: the costs of training and absorbing these new people into the team, he warned, might well exceed the benefits. Brooks' comments were made in the context of traditional, "closed-source" software projects. How does his warning translate to the new world of free/libré open source software (FLOSS) projects? FLOSS projects are completely dependent on volunteer labor; as such, the vitality of a project depends on it's ability to attract, absorb and retain developers or face stagnancy and failure. How do FLOSS projects absorb newcomers without falling afoul of Brooks' man-month paradox? FLOSS projects uniquely make their source code freely available even to outsiders (not just developers— by developers we refer to people who have write access to the source code repository). In addition, this process in most cases begins with mailing list participation. FLOSS projects have mailing lists on which public discussions (open to anyone) concerning the engineering and design of the project are conducted. Outsiders can read the source code, *educate themselves, on their own time*, and join the discussion. Through sustained interest and contributions to the technical discussions, outsiders win the trust of the project's inner circle, and attain developer status along with the keys to the project's source code repository. This process (which we refer to as *immigration*) is unique to FLOSS projects, and is in fact a critical element of the phenomenal success of the FLOSS process. It has been the subject of study by many researchers [7, 10, 11, 19, 26].

[1]**This is a substantially revised and extended version of a paper published at the MSR 07 workshop**. We have added several new charts support our theorizing in section 3, expanded our analysis to consider project differences in detail, revised the analysis to consider the numerical effect of higher patch submissions, and extended the threats to validity section. With this revised submission, we hope to both support the workshop format of MSR 07 as well as create a revised, extended complete archival publication.

Contributions:. Existing work has largely been qualitative in nature. Our goal is to *quantitatively evaluate the influence of various factors* on FLOSS immigration. In this paper, *we develop a theory of FLOSS immigration*, considering three (conflicting) relevant factors that influence if/when a FLOSS participant becomes a developer: *technical commitment*, which is difficult to sustain, and naturally wanes with time, *project-specific skill level*, and *social status*, both of which increase with time. We expect these conflicting factors to cause the rate of newcomer immigration to vary non-monotonically with tenure. We also expect that technical commitment and social status will show a significant effect on the chances of becoming a developer. Second, *we present a quantitative evaluation* of this theory, using statistical hazard-rate modeling, with all available relevant data collected over the entire record of 3 FLOSS projects: the Apache web server, Postgres, and Python. The interesting new findings in our case studies are:

1. The immigration rate, as a function of time spent on the mailing list, is not always steady, or even monotonically increasing (or decreasing). (see Figure 4).
2. Social status, patch submission behavior, and message-sending behavior all influence the immigration rate positively.
3. There are differences between projects on the effects of these factors that could be reasonably attributed to institutional norms of these projects.

Our work has implications for both FLOSS managers and potential FLOSS immigrants. Quantitative modeling can provide useful information concerning the relative effects of different types of behaviors on the likelihood of immigration. For aspiring immigrants, a quantitative study provides information concerning the requirements of being admitted to developer status. In classic GQM style [1], FLOSS project leaders and managers can use this type of approach to evaluate if project behavior is consistent with project needs & norms, and adjust either (or both) as appropriate.

2. Background

In this section, we present related work and then describe the conceptual framework of our approach.

2.1 Related Work

The immigration process in FLOSS has naturally attracted a great deal of attention from researchers. Prior research has analyzed the *attraction* of new immigrants to projects, *barriers* to their entry, and the *process* by which they join the project,

Several papers have analyzed the reasons why FLOSS projects attract newcomers (see, among others *e.g.,* [16, 19, 20, 25]). Suggested motivations include personal need for the software, reputation-seeking, and altruism. There is, however, a serious barrier that curbs these motivations. Programming is a highly knowledge-intensive activity. Even experienced software engineers spend a substantial amount of time building the specific skills needed to execute particular development tasks. This has been noted in numerous studies [5, 8, 21]. A well-known case study by Sim & Holt of immigrants in a *traditional* software project noted knowledge barriers to entry and the importance of mentors [24]. This study also noted the need for a "minimal interest match" between a new immigrant and the project. In FLOSS projects, the immigrants self-select for interest, and voluntarily overcome the skill barrier.

Barriers notwithstanding, large, popular projects such as the Apache web server attract a large amount of volunteer labor. In fact, many of the larger FLOSS projects have, to varying degrees of formality, developed processes that regulate the admission of new immigrants into full developer status. This process, also called a *joining script* in the literature, has been studied in the context of a few OSS projects. Von Krogh, *et al* have studied several aspects of the immigration process in the Freenet FLOSS project [26]. They used data gathered from interviews, publicly available documents such as FAQs, email archives, and versioned source code repositories. They found that certain types of email actions, such as offering bug fixes, are much more common among newcomers who eventually become developers. They also note that the locus of the first development activity by immigrants is strongly determined by modularity, complexity, *etc.* of the target file or class. Lastly, newcomers' first contributions are specialized according to their prior skills. In an ethnographic study, Ducheneaut examined the Python project and the interactions of a particular individual as he transitions from a newcomer to a full-fledged developer [11]. He found that prior technical activity and social standing in the community are strong indicators of the likelihood of achieving developer status. Gutwin, Penner, and Schneider studied group awareness for distributed OSS projects [14]. They found that communication in the form of mailing lists, text chat, and commit logs were the primary media from which awareness was drawn. They noted the importance of these tools in keeping an OSS project organized. Jensen and Scacchi [17] documented in detail the different roles in several open source projects, and the processes by which people progress through the roles. Herraiz *et al* [15] presented quantitative descriptive statistics on the rate of immigration in the GNOME project, and observe some differences between paid and volunteer developers, but do not do any statistical modeling of the predictors of immigration behavior.

In this paper, we also study immigration, but using *a quantitative approach*, based on hazard rate analysis. Hazard rate analysis, or survival analysis [9], is a well-established method used to study time-dependent phenomena such as mortality, artifact failure, recidivism of convicts, employment durations, business failures, *etc.* Using statistical models, one can estimate the influence of time and other predictors on the occurrence of expected events (*e.g.,* duration since surgery, prior smoking history, diet, chemotherapy, *etc.* on cancer patient mortality). We introduce the use of this technique to study the immigration into FLOSS

projects; specifically, we study the duration from the first appearance on the mailing list of an individual, to the time the first commit, if any, is made by that individual. Details of this technique are presented later; first, we develop the conceptual framework and the hypotheses of interest.

2.2 Conceptual Framework

In a departure from previous research, this paper considers how the likelihood of becoming a developer varies with tenure in a community, and also quantitatively evaluate the importance of factors such as social status and demonstrated technical skill. We begin with a conceptual framework for the mechanisms that influence the attainment of developer status. This conceptual framework directly leads us to the phenomena we model as predictor variables in the statistical hazard rate model. It also helps us theoretically explain the observed non-monotonicity in the hazard rate (as will be seen later).

We consider four different factors that influence acceptance into developer-hood.

- *Technical commitment to project:* how committed is the developer to the success of this project? How long does s/he sustain that technical commitment?
- *Skill Level:* How knowledgeable/skillful is this developer relative to this specific project?
- *Individual Reputation:* What is the status of the individual in this community?
- *Project Specificity:* Are there significant differences in different projects, relating to immigration?

To become a developer, a individual must both acquire project-specific technical skills; *and then* s/he must win the community's trust by demonstrating these skills, via email participation and by contribution of work products. This takes commitment. Therefore, the developer needs to make a long-term commitment to first acquire those skills, show them to the community, and earn their trust. Experienced software engineers are well aware of the effort required to sustain specialized technical skills relevant to an evolving system for long periods of time.

2.2.1 Commitment

A developer's commitment to a project will arguably decay with time, increasing the likelihood that a given person will quit. Sustaining working skills & knowledge in a large, complex project is a formidable undertaking, and unpaid volunteers who have not yet reaped the professional reward of being admitted into the inner circle cannot be expected to keep up their effort for too long.

This effect can be expected to be somewhat attenuated for people who become developers, since we can expect that these people have made an invested effort to earn that privilege, and have developed valuable relationships within the community.

To the variation of commitment with time, we examine how many different non-developers are active during each month since their first appearance on the mailing list. While all of these are potentially candidates to become developers, prior research shows that *patch submitters* are the most technically engaged in the community, and most likely to become developers(See von Krogh [26]).

Figure 2 shows the number of non-developers who remain active on the mailing list in Postgres (and similarly in all the projects we studied) decays steadily as tenure increases to the maximum (note that y-axis, counts, is log-scale) around roughly the 100^{th} month. (*i.e, lifetime of the email archive*). In contrast, after a much shorter tenure interval of 40 months, there are very few active *patch submitters* remaining. All three projects we considered show the same pattern (though with different time periods). in Apache, this period is a little longer, about 50 months, and in Python, it's much shorter: very few email participants submit any patches after their 10^{th} month.

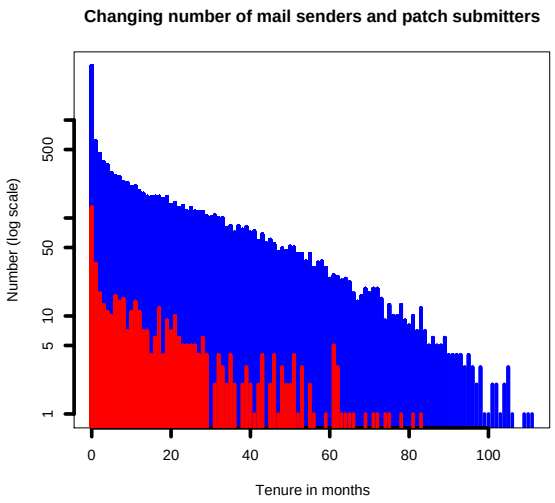

Figure 2. Variation in number of non-developers (log scale) who actively submit emails (top curve) and patches (bottom curve) in Postgres with tenure in months

2.2.2 Knowledge and Skill Level

As a developer spends more and more time on a project, she can be expected to gradually become more knowledgeable and skillful. This difficult, time-consuming process of learning the details of a specific system and development environment (sometimes known as *discovery*, or *ramp-up*) is documented by prior research [8, 24]. In many cases, even the initial email is sent by an individual on the developer mailing list only after some initial study; quite often people submit patches during their first month of activity on the mailing list. In the Postgres project, we find that the median time for first patch submission is during the second month of mailing list participation. For Apache, the median time is the second month, and for Python, the median time is the sixth month.These numbers indicate the time commitment required for skill acquisition.

2.2.3 Individual Reputation

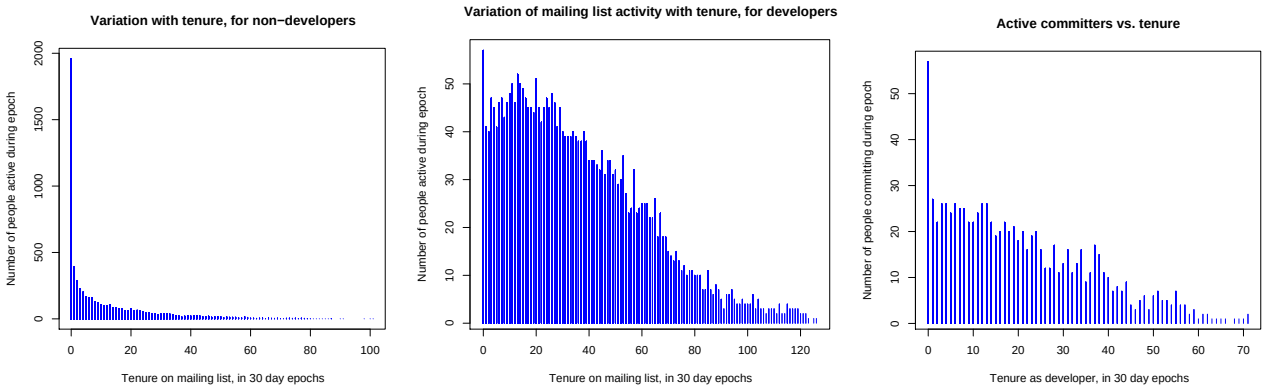

Figure 1. All figures (for the Apache project) have tenure in 30 day epochs on the x-axis, and indicate number of active people during that epoch of their tenure on the y-axis. Left most figure shows email activity for non-developers, middle email activity for developers, and rightmost shows file commit activity. Python and Postgres show similar patterns. Number of active people declines in every case, indicating gradual waning of engagement

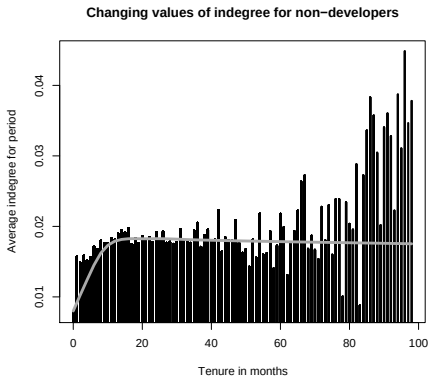

Figure 3. Variation in number of respondents (in-degree on the social network) with tenure on the mailing list. Note that the data beyond 60 months is based on fewer than 10 or so participants, and is thus unstable. Grey line shows smoothed trend. Notice initial rise, following by flat/declining slope over time.

An individual's reputation can be expected to increase with tenure and activity on the mailing list. Prior research has documented the need to build community reputation before being admitted as a developer [11, 26].

Social network theorists have developed validated network measures of community importance, based on the network of interactions [27]. In the networks for our analysis, each node or actor represents a mailing list participant. If actor A posts a message and actor B responds, then there is indication that B had some interest in A's message. Therefore, we create a directed tie from B to A.

Social network metrics include in-degree, out-degree and other measures. In-degree and out-degree are defined as the number of ties directed towards and away from an actor respectively. If A has high in-degree, that indicates that many people found A's messages of interest, and thus that information contributed by A is relevant and interesting. High out-degree indicates that A finds many people's messages of interest.

In figure 3, we show the variation of median in-degree for non-developers (who form the candidate pool for new developers) with tenure in the Postgres project. It can be seen that in-degree, as a function of tenure, first increases, then flattens until about 40 months, and then decreases until some point when there are so few mailing list participants remaining (with such long tenures) that the data becomes unstable.

The decrease in in-degree can be related to the patch submission data; after around 3-4 years in Postgres, non-developers tend to stop submitting patches and are presumably less technically engaged. A similar phenomenon is observed in Apache and Python: median in-degree for non-developers peaks during the second or third years of tenure, and then declines.

2.2.4 *Project-Specific Considerations*

FLOSS Projects have to a greater or lesser degree developed specific cultural norms on how immigration is handled. These norms are enforced to varying degrees of formality. Much work remains to be done on identifying the precise nature of these cultures, constructing theories concerning the influence of these factors, and validating these theories. We analyze only three different projects. However, these projects have been described by Berkus [2] to be examples of three very different types of projects. Apache is a *foundation* with a well-organized, hierarchical governance structure and formalized policies. Postgres is a *community*, more informal, with consensual group decision making. Python is *monarchist*[2] with an authority figure, Guido Van Rossum. With just 3 projects, quantifying the precise influence of these factors is not possible. We have, however, fortuitously chosen projects with very different cultures, and characteristics. We consider here 3 factors: *structure of governance*, *formalization of immigration* and *technical complexity*.

Foundation: The Apache project[3] arose out of a loose

[2]For a description of these project types, see `http://www.powerpostgresql.com/5_types`

[3]Please see `http://httpd.apache.org/ABOUT_APACHE.html` for details

coalition of developers who were developing variants and patches on the original NCSA web server built by Rob McCool. In 1999, a formal structure (the Apache Software Foundation, ASF) was created to provide an "organizational, legal, and financial support for the Apache HTTP Server". There are now several projects operated by the ASF. The ASF prescribes a well-defined hierarchy of governance with several roles. The most relevant role for us is a *committer*, or a person who has earned the right to commit to the source code repository. Advancement in the ASF hierarchy, as with other FLOSS projects, is based on *meritocracy*[4]. Each project in the ASF, including the HTTP Server is governed by the Project Management Committee (PMC), whose members are chosen from among the committers. PMC members make all the major decisions about the software; specifically, they alone have the right to vote to grant committer-status to people who are actively contributing valuable additions to the project.

PMC members are naturally the elite and most senior people among the committers. Of the 3 projects we studied, Apache has the most well-developed, organized, and documented organizational structure. It also has the most detailed description of the different roles that participants may aspire to, and how these roles may be acquired. Apache is by far the most widely known and influential of the 3 systems we have studied; current data indicates that around 60% of the webservers accessible on the internet are running Apache. One could reasonably speculate that the enormous interest in, and high visibility of this project has led to the mature organizational structure. As far as technical complexity; while the Apache webserver is a large system, it is relatively well modularized; in addition, the basic logic of a web-server is comparatively simple. Our conclusion therefore is that Apache has *strong organizational structure*, a *well-established immigration process* and is of *moderate technical complexity.*

Community: The PostgreSQL project also has two "levels" of developers: the core development group, which someone must be invited into; and the normal contributors with write-access to the repository[5] and other participants to post to the mailing list and submit patches. Acceptance into the committer category requires demonstration of technical expertise. However, there is considerable debate within the community on the exact nature of the process by which committer status is granted. Recently (March 2006), there has been a very interesting thread of discussion on the Postgres developer mailing list [6] on how a contributor to the project is accepted as a committer. In fact, Bruce Momjian, who is a highly central and influential developer in Postgres, states, in response to a query, in the above thread.[7]:

```
> I'm trying to understand the social structure of
> the PostgreSQL  project. Is there a documented
> history  of who became a committer and
> when? That you could point me to?

Uh, we don't have a very formal organization, so
this information doesn't really exist.  MIT did a
study about our open source community. You
might find that useful, but I don't have the URL
```

Later respondents on this thread pointed out that this information could be easily obtained (as we did) by combing through CVS logs to find out when someone first made a commit; however, the comment regarding the lack of a formal organization from a core developer is telling. Later in the same thread, the participants discuss how contributing patches can help earn committer-status. The comments make it clear that the threshold is very high: a participant gains committer status when they become so productive that core developers give in, and let them do their own commits. Again, quoting Momjian: "It is a case where the volume of patches just overwhelms us and we give them commit access". Finally, Postgres is technically complex: it provides flexible, concurrent data access, at very high transaction rates, to an extremely complex set of secondary storage data structures using highly refined data-structures, concurrency, and query optimization algorithms, *etc.* These features and technologies interact and cross-cut, leading to very complex software indeed. In conclusion, PostgresSQL has an *informal organizational structure*, *an informal, but highly demanding and selective immigration process* and is *technically very complex.*

Monarchist: The Python project appears to be the most informal of the three. There is no stated policy for accepting contributors into the developers circle. In typical Python fashion, a rather tongue-in-cheek description of the road to developer-hood is given in "The school of hard knocks"[8] by Raymond Hettinger, a core Python developer. This document does not specify an actual process for gaining developer status, but rather suggests several tips to making ones self visible to the core developers. Suggestions include writing Python Enhancement Proposals (PEPs), submitting documentation and unit tests with patches, following code conventions, and "submitting great ideas" (with implementations being an added bonus). Folklore concerning the project indicates that Guido Van Rossum, the founder of the project exerts a very high level of authority and influence (unusual among FLOSS projects) He is the self-anointed BDFL (benevolent dictator for life) of the Python project. Technically, Python is an interpreter for a language, and has a well-modularized structure. In summary, Python has a *very informal but very centralized organizational structure*, has *no stated immigration policy* and is of *moderate technical complexity*

Summary. We hasten to point out that these are informal observations based on available documents. We certainly do not have sufficient formality, data, or testable theories of how these factors would influence immigration, nor do we have a sufficient population of projects to analyze data. However, these factors can be expected to influence immi-

[4]See `http://www.apache.org/foundation/how-it-works.htm#meritocracy`

[5]`http://www.varlena.com/GeneralBits/74.php`

[6]`http://archives.postgresql.org/pgsql-advocacy/2006-03/msg00053.php`

[7]Momjian in the quote below presumably refers to Lakhani's work, as indicated in a follow-up email in the same thread.

[8]`http://mail.python.org/pipermail/python-dev/2002-September/028725.html`

gration, and it will be important to bear these in mind while interpreting the results of the hazard rate models for each project. During the discussion of the hazard models, we will refer to these factors as possible influences. Further study is clearly required.

In the next section, we present formally the hypotheses that the subject of our study.

2.2.5 Hypotheses

We surmise that the likelihood of acceptance into the core, elite developer group of an open source project is likely to be modulated by three effects: commitment, skill acquisition/demonstration, and reputation. For every individual there is a "race" going on: will s/he become skilled and reputable enough to become a developer before s/he loses interest? In some sense, there is a race within each individual's tenure time line to acquire the requisite skill set and reputation before commitment wanes. We therefore expect the following:

Hypothesis 1 *Likelihood of attaining developer status will rise with tenure, peak at some point, and then decline.*

Hypothesis 2 *Demonstration of skill level, such as patch submissions, will increase the likelihood of becoming a developer.*

Hypothesis 3 *Social status will positively influence attainment of developer status.*

We conclude this section by noting that non-monotonic rates of event occurrences, which grow with time, and then decline (or vice versa) are observed in other settings. Divorce rates in marriage tend to be high initially, and decline before increasing again. Fichman & Levinthal [12] describe "the liability of adolescence" in the case of employment duration where new hires tend have an initial honeymoon period, after which they are at greater risk of job dissatisfaction; if they survive this period, skill acquisition may lead to improved job performance and satisfaction. Fichman & Levinthal argue that this phenomenon explains a non-monotonic rate of job changes. Katz [18] describes a related phenomenon whereby employees go through phases of *socialization* with increasing skills and connections, *innovation* with relatively high productivity, and *stabilization* of steady-state or decline.

3. Analysis

In this section, we present our data extraction methodology, some background on the statistical models used, and the results.

3.1 Data Extraction & Cleaning

We gathered source code repository information (who changed what file and when?) and email archive information (who sent an email? who replied to it? when?) in a manner similar to previous research [13, 22]. Extra effort was spent to ensure that email aliases and repository author identities were properly resolved, using automated and manual methods [3]. We built a social network from the email correspondence and computed social network measures [27] on a monthly and cumulative basis. we expect that social network positions/measures would be indicative of social status, and thus of likelihood of attaining developer status. We also extracted patch submissions from emails and searched the project repository for evidence of at least partial patch application[9]. Prior research has indicated the importance of patch submission in gaining developer status [11]; so we expected that this data would be an important predictor.

For each developer, the transition interval is the time between their first appearance on the mailing list and their first commit to a file. This interval is essentially the "response variable" we are trying to model statistically, in order to shed light on the factors affecting time interval until immigration.

3.2 Predictors & Univariate Statistics

All the variables used in our study are gathered monthly for the complete population of potential immigrants "at risk" (i.e. all mailing list participants who are not yet developers). Each record described below is for one email participant, for one month. In each case, n is the number of records, c is the number of candidates, and i is the number finally immigrating.

First, based on previous research, we conjectured that patch submission is important; the variable *patches_sub* indicates the number of patches submitted by this individual. Second, as discussed earlier, we expect that *norm_indegree* (normalized in-degree), as a measure of the degree of response/interest to this individual is important. This is measured as the proportion of the total population that has responded to this candidate since his/her first post. The variable *sent_cum* measures the total number of messages sent by this individual prior to this month. Finally, *devs_cum* is the total number of developers in the community. This is used as a control variable which allows us to control for the effects arising from size, such as greater openness to immigrants in smaller projects (with fewer developers) seeking to attain critical mass. Another control variable, *time_trend*, is simply calendar time in years (as opposed to the tenure time, which begins for each person with their first observed email) to control for unobserved effects relating to project age.

It should be noted that Apache and Postgres have much longer histories; the email list for both is available for over 10 years. Python is shorter with 7 years worth of email data available respectively. Although Apache's social network has been building for well over a decade, the source code repository that we used (which contained the data for the 2.0 version of the Apache web server) only con-

[9]We encourage the reader to see our *Mining Software Repositories 2007* paper located at `http://wwwcsif.cs.ucdavis.edu/~bird/papers/bird2007dps.pdf` for details of this process

tained data for 5 years. As a result, we only consider those who joined the mailing list (mailed for the first time) *after* the source code repository became available, on January 5, 1999. There are 1,445 such individuals.

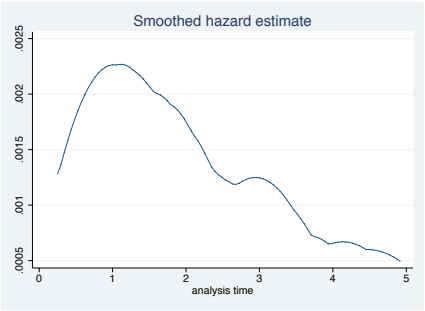

Figure 4. Fitted hazard rate estimate for immigration events in Postgres with time scale in years

After gathering the variables, we accounted for multicollinearity between variables by checking correlations. We omit these details here for brevity [10].

Variable	Mean	Std. Dev	Min	Max
(**Postgres**) n = 178,510 c = 3,545 i = 20				
patches_sub	.010	.42	0	90
norm_indegree	.0013	.0082	0	.31
devs_cum	15.70	4.21	1	21
sent_cum	13.52	67.76	1	2724
(**Apache**) n = 50,170 c = 1,445 i = 30				
patches_sub	.036	.78	0	58
norm_indegree	.0015	.0088	0	.238
sent_cum	7.34	18.18	1	273
devs_cum	46.32	9.76	3	57
(**Python**) n = 45,216 c = 1,320 i = 62				
patches_sub	.0072	.31	0	46
norm_indegree	.0039	.023	0	1
devs_cum	58.69	14.21	0	79
sent_cum	13.88	54.45	1	1648

Table 1. Univariate Statistics of the predictors

Figure 4 is a plot of the raw data that indicates how people immigrate in the Postgres project. This shows the rate of participants becoming developers on the y-axis versus tenure time in years on the x-axis. Notice the non-monotonicity of the curve as it increases in the rate until a peak around one year followed by a gradual decrease until five years. This curve is generated by smoothing the raw data. We present the actual statistical model later. More figures, for our 3 subject projects are available at `http://macbeth.cs.ucdavis.edu/hazard`

3.3 Hazard Rate Analysis: background

Hazard rate analysis, or *survival* analysis [4, 9], can model stochastic time-dependent phenomena such as systems failure, cancer survival, employment duration , business failure, etc. We use survival analysis not to *predict* who will become a developer or when, but rather to *understand* which factors influence, to what degree, the duration and occurrence of such events (*e.g.*, gender, age of onset, smoking history for a cancer survival analysis). If a model has a statistically significant fit, then the details (based on the estimated coefficients of the predictors) shed light on hypotheses concerning the effects of the predictors on the survival. It is thus a natural method for a quantitative study of the process by which a mailing list participant immigrated to become a developer in a FLOSS project.

We now informally present some background for the hazard rate model (see [4] for details) The hazard rate function captures the *rate* at which events of interest occur, and models their dependence on time and the other predictor variables. First, we present a definition of the hazard rate, and its dependence on the time variable t. Suppose the event does not occur until exactly time T. The *hazard rate* of the event of interest is the probability of the event occurring in an infinitesimal time interval δt starting at time t (given that it hasn't occurred until then) divided by δt. It is modeled as a *hazard rate* function $\lambda(t)$:

$$\lambda(t) = \lim_{\delta t \to 0} \frac{P(t \leq T < t + \delta t \mid T > t)}{\delta t}$$

Consider the probability of "survival", that is, the probability that the event has not occurred yet. Assuming that the event actually occurs at time X, the probability that X is later than t is given by:

$$P(X > t) = exp(-\int_0^t \lambda(s)ds)$$

The simplest possible model is that the rate is a constant, λ_c, which gives rise to the common exponential model for survival, with survival becoming always less probable as time increases:

$$P(X > t) = exp(-\lambda_c t)$$

The problem with this simple model is that it doesn't allow for modeling multiple predictors (*e.g.,* age, gender, ethnicity, profession) or non-monotonic rates of failure. A general class of models, called the *proportional hazards* model, allows the introduction of other predictors (besides time). Such models in general are written as:

$$\lambda(t) = f(t) * g(\mathbf{x})$$

where f is purely a function of time, $\mathbf{x}$ is a vector of predictors, and g is an appropriate function. In our setting, we seek to investigate a) if there is a non-monotonic dependence on tenure, and b) if social factors and prior evidence of knowledge and skill have an effect. For our purposes, the easiest model to use is the *piecewise constant exponential hazard rate model*. We assume that the purely time dependent part $f(t)$ is fixed over each. Thus,

$$f(t) = exp(\alpha_{tp_k}) \text{ for } t \in tp_k, \text{ where } tp_k = (c_{k-1}, c_k]$$

and the intervals $(c_{k-1}, c_k]$ are chosen to cover the duration of the available data, and α_{tp_k}, one for each interval, are constants. This gives us the flexibility of seeing if the data supports the hypothesis that these rates change non-monotonically; in fact any pattern of increasing/decreasing

[10] More details are available at `http://macbeth.cs.ucdavis.edu/hazard`

rates is possible. For the parametric part $g(\mathbf{x})$, we use an exponential model, thus:

$$\mathbf{g}(\mathbf{x}) = exp(\mathbf{b} \cdot \mathbf{x})$$

where $\mathbf{b}$ is a vector of parameters derived by statistical fitting. This allows us to examine the influence of predictors such as prior patch submission history, social network status, *etc.* on the time to immigration.

3.4 Results

As explained above in section 3.3, the hazard rate function is modeled in this form:

$$\lambda(t) = exp(\alpha_{tp_k}) * exp(\mathbf{b} \cdot \mathbf{x})$$

The first part (previously denoted $f(t)$) is a constant for each time interval tp_k, and the second part depends on the vector of variables $\mathbf{x}$. We fit this model for each of the mined projects. During the analysis, we first started with a baseline model, consisting of the piecewise time periods, and the control variables (*devs_cum* and *time_trend*). We then added the variables *patches_sub* and *sent_cum*, which have to do with an individual behavior and skill level. Finally, we added the *norm_indegree* variable, which represents the community response. For each of these 3 steps, we built the piecewise constant proportional hazard rate model and calculated the likelihood ratio χ^2 to judge the improvement in fit. The results are shown in table 2. It can be seen that the improvement in fit is highly significant in the first step (two degrees of freedom) as well as the second (one degree).

Variable Added	Apache Model LR χ^2	Postgres Model LR χ^2	Python Model LR χ^2
sent_cum, patches_submitted	144	62	52
indegree	10	56	30

Table 2. Improvement in fit (likelihood ratio χ^2) provided by variables in the model. The base model included the time periods, and control variables *devs_cum* and *time_trend*. All χ^2 values are highly significant. All variables were checked for statistical independence

The results from the final model in each case are shown in table 3. The table shows one variable in each row, with coefficients and their significance in each column, separated out for each project. We show the estimated values of all $\alpha_{tp_k}s$ and the components of the vector $\mathbf{b}$; the size of these coefficients represents the size of the effect of the variable, and Z score represents the statistical significance of the estimated value. The Z score is calculated by dividing the estimated value by the standard deviation (not shown). The probability (next column) is the likelihood that the coefficient is actually zero in the proportional hazards model (*i.e.* the variable has no effect on the rate, since e^0 is 1). The lower the probability, the more statistically significant the result. Generally, values less than 0.05 are considered statistically significant. The absolute values of the coefficients are quite different because the range of values of the relevant variables is different. The actual effects can be judged by considering both the value range and the coefficient, as we illustrate below.

We first discuss the effect of variables that are dependent on the individual. These variables (indicated with *) are located at the lower part of the table. Their coefficients are to be interpreted as *log(proportional effect)* on the hazard rate for unit change in the value. For example, *patches_sub* is a variable that indicates the number of patches submitted by a particular mailing list participant. This effect is largest in Python, where each submitted patch increases the hazard rate by a factor $e^{0.093}$, or about 9.7%. In Postgres, this effect is about $e^{0.054}$ or an increase of 5.5%. The effect is positive, but only significant at about the 0.1 level in the Apache project. *Sent_cum* is the total number of messages sent by an individual. It's effect is significant in Apache resulting in a $e^{0.021*18.18}$ (1.5 fold) increase in the hazard rate for one standard deviation increase in value. The rates of increase are statistically significant, but more moderate for the other projects; $e^{0.003*54.44}$ (18%) for Python and $e^{0.002*67.76}$ (15%) for Postgres for their respective standard deviations. The social network measure *norm_indegree* is statistically very significant in all 3 models; however, the effect varies per project, increasing the rate by about 62% in apache, 77% in Postgres and 22% in Python for one standard deviation increase in normalized indegree.

Turning to the time dependent variables in the hazard rate model we show two classes of variables: the first class, $tp1$ etc., constitute the pure time dependence of the hazard rate model. The proportional effect of the rate function can be interpreted as:

$$e^{\alpha_{tp1}*tp1} * e^{\alpha_{tp2}*tp2} * e^{\alpha_{tp3}*tp3} * \ldots$$

Each variable $tp1 \ldots tp6$ should be interpreted as binary, taking on a value of 1 while t is in that period, and 0 otherwise. Thus during time period $tp1$, the proportional effect on the rate fun is simply $e^{\alpha_{tp1}*tp1}$. The absolute value of the rate resulting from this is quite low, in both Apache and Postgres reflecting the low base rate at which people join the project. In both cases where the fit is significant, we see an increase and then a decrease in the hazard rate. The piecewise-constant time-dependent part of the hazard rate model does not show a statistically significant affect on the hazard rate in Python. We believe that due to the centralized nature of the governance structure and more informal immigration process in Python, visibility (in terms of patches submitted and mails sent) plays a much larger role than tenure within the community.

Finally, we control for two potentially confounding variables: in order to control for effects arising from project age, *time_trend* measures the age of the email archives in years. This is different from $tp1 \ldots$, which are tenure periods per person. The model shows a negative effect in Postgres and Python. The total number of developers, *devs_cum*, is used to control for the size of the population who play a central role in deciding if someone becomes an immigrant. This has a positive effect in Python and Postgres. The reasons for the effects of these variables is not clear to us presently, but are not concerns since they are used for control and not prediction, though they do present avenues for future study. Neither of these control variables had a statistically significant affect in the Apache project; we believe this may have roots in the more formal norms associated with *foundation* style projects, but we cannot conclusively affirm this.

Variable	Apache Model			Postgres Model			Python Model		
	Coef.	z	P >|Z|	Coef.	z	P >|Z|	Coef.	z	P >|Z|
tp1 (0-6 months)	-6.04	-3.10	0.002	-4.14	-2.12	0.034	1.60	1.00	0.315
tp2 (6-12)	-5.40	-2.74	0.006	-4.55	-2.29	0.022	1.80	1.05	0.293
tp3 (12-24)	-5.89	-2.83	0.005	-3.42	-2.25	0.024	1.59	0.96	0.339
tp4 (24-36)	-8.04	-3.35	0.001	-3.95	-2.68	0.007	1.20	0.72	0.469
tp5 (36-48)	-7.05	-3.09	0.002	-5.68	-2.52	0.012	1.12	0.63	0.530
tp6 (48-60)	-7.40	-2.69	0.007	-6.68	-2.65	0.008	0.89	0.42	0.677
norm_indegree*	54.68	3.10	0.002	69.37	6.21	0.000	8.54	4.57	0.000
patches_sub*	0.053	1.61	0.108	0.054	2.04	0.041	0.093	3.86	0.000
sent_cum*	0.021	5.64	0.000	0.0021	2.13	0.033	0.0037	3.46	0.001
devs_cum	0.099	1.41	0.159	0.95	2.29	0.022	0.11	2.46	0.014
time_trend	-0.56	-1.22	0.223	-2.26	-2.49	0.013	-1.36	-3.11	0.002

Table 3. Results of Hazard rate model fit. Coefficients represent log-proportional effect of the relevant variable on the hazard rate. For example, in the case of Python, submission of one additional patch (patches_sub) increases the rate by $e^{0.093}$, or nearly 10%. $tp1$, $tp2$, *etc.* are time periods with ranges marked in months. Note that in Postgres and Apache the rate increases and then decreases as we move through from $tp1$ to $tp4$. Since there are very few non-developers that stay past 4 years, the interpretation beyond this point is unclear.

Summary: The model results for the three projects share a fair amount of similarity in terms of the direction and degree of effects that the predictors have on the immigration rate. We draw some conclusions from the fit of these models with regard to our hypotheses below.

Non-monotonic tenure dependence: In both Apache and Postgres, the models support the hypothesis that the hazard rate increases, and then decreases. In all three cases, the data, when plotted, shows this trend; however, in Python the results are not statistically significant. The difference may be due to the centralized community structure and more ad hoc immigration policies in this *monarchist* project or could be attributable to the calendar duration of the projects: Python is 4 years younger than both Apache and Postgres; perhaps the community's reaction to newcomers is still evolving. Thus, we conclude that *Hypothesis 1 is supported in Apache and Postgres, but is indeterminate in Python.*

Patch submission effect: In Python and Postgres, prior history of patch submission has a significant effect, with each patch submitted increasing the hazard rate by 10% and 5% respectively. The effect is positive and within the same order of magnitude, but not statistically significant in Apache. We thus conclude that demonstrated skill level via patch submission plays an important role in Python and Postgres, but results are inconclusive in Apache. The effect in Python is especially strong (0.093 for Python, vs 0.053 and 0.054 for Apache and Postgres) . This is consistent with stated institutional norms of the Python project, which emphasize display of skills through patch submissions and other technical contributions as a way of gaining status. *Hypothesis 2 is supported in Python and Postgres, but not in Apache.*

Social Status/Activity: In all three models, the social network measure, indegree, which is a measure of the breadth of response to an individual within the community has a significant effect, although the effect is moderate. This indicates that community response to participants does play a role in developer immigration on each of the projects studied. This is especially interesting given the varied governance structures and levels of formality with regard to the immigration process in the projects. The effect of social status (*norm_indegree*) is specially strong in Postgres, reflecting Josh Berkus's description[11] of Postgres as a *community* project, where decisions are made communally. Still, the significance in all projects indicates a phenomenon that may generalize well to a significant portion of other OSS projects. So we conclude that *Hypothesis 3 is supported in all three projects.*

4. Threats to Validity

We now discuss possible threats to validity and explain how we address them (when possible).

4.1 Internal Threats

Our patch and social network data are extracted from the project development mailing lists. We may be missing some data if participants interact on IRC channels, via direct email or in other ways (even face to face in some instances). This method is justified for a few reasons. Current research literature [3, 11, 16, 22] suggests that patch submissions and community discussions occurs on developer mailing lists. Second, accepted open source tradition (and policies within many FLOSS communities) indicate that the developer mailing list is the standard place to submit patches, discuss the software, and for newcomers wanting to contribute, to interact with the community[12]. One reason for this is that the mailing lists have the highest visibility of the community communication media. As the mailing lists are the main form of communication in FLOSS communities, we believe that interactions via other mechanisms will be manifest as similar interaction on the mailing lists.

4.2 Threats to Construct Validity

One of our hypotheses is that community perception of a participant's technical skills and knowledge has an effect on becoming a developer. Our method of measuring perceived technical skill is by examining the number of patches submitted and accepted into the source code repository. While there is accepted literature [23] that supports the notion that contributing patch "work-gifts" is one the best ways

[11] See `http://www.powerpostgresql.com/5_types`

[12] please see `http://httpd.apache.org/ABOUT_APACHE.html#Development`

to exhibit technical skill, there are other ways as well. We currently do not capture and quantify technical discussion of software architecture or other types of messages that may improve community perception of an individual's skills.

4.3 External Threats

There are limitations to how well these results may generalize to other FLOSS communities. As part of our criteria for analysis, we needed projects with a long history of public archived data. We also selected projects with a large community of participants and developers so that we would have enough observations to conduct statistical analysis. These criteria necessarily introduce some bias into our results, and so they are likely only relevant for similar projects. Since we have the tools and theoretical infrastructure, we hope to test our hypotheses on other projects.

4.4 Threats to Conclusion Validity

One threat to conclusion validity in any statistical analysis is the assumptions made in the model. If the assumptions required for the model are not met by the data then the results of the analysis may appear to be quite significant when in fact, they are not. We avoid this abuse by using a very flexible model of duration dependence. Because the model is piecewise, we are not constraining the shape of the hazard rate curve. This model makes no assumptions about the *a priori* influence of tenure on the process being modeled. In the model, we are controlling for a number of variables such as the time elapsed and the number of developers in the model at each month. There may be other phenomena that we are not controlling for which could affect our results.

One concern might be the limited number of successful immigrants; however, hazard rate models are well-suited for situations with extremely low rates of failure, and are used in failure rate analysis in system design, death rates in populations, etc.

5. Conclusions

We mounted a quantitative study of the immigration process in FLOSS projects. We hypothesized that 1) the rate of immigration is non-monotonic 2) demonstrated technical skill has an impact on the chances becoming a developer 3) social reputation also has an impact on becoming a developer. We mined data from three FLOSS projects with differing governance structures, levels of immigration formality, and technical complexity: the Apache web server, Postgres, and Python projects. This data was studied by using a piecewise-constant proportional hazard rate model to estimate the effects of predictors associated with on developer immigration rates. Each of these hypotheses is supported by at least two of the three projects under study.

6. References

[1] V. Basili, G. Caldiera, and H. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, 1:528–532, 1994.

[2] J. Berkus. The 5 types of open source projects. March 20, 2007 http://www.powerpostgresql.com/5_types.

[3] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining Email Social Networks. In *Proc. of the 3rd International Workshop on Mining Software Repositories*, 2006.

[4] H. Blossfeld and G. Rohwer. *Techniques of event history modeling*. L. Erlbaum Mahwah, NJ, 1995.

[5] F. Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, 1987.

[6] F. Brooks. *The mythical man-month*. Addison-Wesley, 1995.

[7] A. Capiluppi, P. Lago, M. Morisio, and D. e Informatica. Characteristics of open source projects. *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, pages 317–327, 2003.

[8] T. Corbi. Program Understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.

[9] D. Cox and D. Oakes. *Analysis of survival data: Monographs on Statistics and Applied Probability*. Chapman and Hall, 1984.

[10] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[11] N. Ducheneaut. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368, 2005.

[12] M. Fichman and D. Levinthal. Honeymoons and the Liability of Adolescence: A New Perspective on Duration Dependence in Social and Organizational Relationships. *The Academy of Management Review*, 16(2):442–468, 1991.

[13] M. Fischer, M. Pinzger, and H. Gall. Populating a Release History Database from Version Control and Bug Tracking Systems. *Proceedings of the International Conference on Software Maintenance*, 2003.

[14] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 72–81, New York, NY, USA, 2004. ACM Press.

[15] I. Herraiz, G. Robles, J. Amor, T. Romera, and J. Barahona. The processes of joining in global distributed software projects. *Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 27–33, 2006.

[16] G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7):1159–1177, 2003.

[17] C. Jensen and W. Scacchi. Role migration and advancement processes in ossd projects: A comparative case study. In *Proceedings, ICSE 2007*, 2007.

[18] R. Katz. Managing professional careers: the influence of job longevity and group age. In *Managing strategic and innovation and change: a collection of reading*, pages 0–19. Oxford University Press, 2004.

[19] K. Lakhani and R. Wolf. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *September, MIT Sloan Working Paper No: 4425–03*, 2003.

[20] J. Lerner and J. Tirole. Some Simple Economics of Open Source. *Journal of Industrial Economics*, 50(2):197–234, 2002.

[21] S. Letovsky. Cognitive processes in program comprehension. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 58–79, Norwood, NJ, USA, 1986. Ablex Publishing Corp.

[22] A. Mockus, J. D. Herbsleb, and R. T. Fielding. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, July 2002.

[23] E. Raymond. Homesteading the Noosphere. *First Monday*, 3(10), 1998.

[24] S. Sim and R. Holt. The Ramp-up Problem in software projects: A case study of how software immigrants naturalize. *20th Int. Conference on Software Engineering*, pages 361–370, 1998.

[25] E. von Hippel and G. von Krogh. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science*, 14(2):209–223, 2003.

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

[27] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.