Open Borders? Immigration in Open Source Projects

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, committed foundation of the project, selected through a meritocratic process from a larger number of people who participate on the mailing list. Most projects carefully regulate admission of outsiders to full developer privileges; some projects even have formal descriptions of this process. Understanding the factors that influence the "who, how and when" of this process is critical, both for the sustainability of FLOSS projects, and for outside stakeholders who want to gain entry and succeed. In this paper we mount a quantitative case study of the process by which people join FLOSS projects, using data mined from the Apache web server, Postgres, and Python. We develop a theory of open source project joining, and evaluate this theory based on our data.

1 Introduction

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. 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. Through sustained interest and contributions to the technical discussions, outsiders become trusted, attain developer status along with the keys to the project's source code repository. This process in fact a critical element of the phenomenal success of the FLOSS process. It has been the subject of study by many researchers [4, 7, 8, 13, 20].

In this paper, we follow previous researchers that have studied immigration success stories in FLOSS projects, using a quantitative approach based on extensive data mining. We build upon existing work by Ducheneaut [8], von Krogh [20], and others to:

- *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. As a result, we expect a non-monotonic rate of newcomer immigration, *viz.*, first an increase in the rate, and then a decrease. We also expect that technical commitment and social status will show a significant effect on the chances of becoming a developer.
- *present a quantitative evaluation* of this theory, using statistical hazard-rate modeling, with data mined from three case studies: the Apache web server, Postgres, and Python. From this quantitative evaluation we find insights not apparent in previous, purely qualitative evaluations, for instance that the immigration rate is non-monotonic (see Figure 1).

2 Background

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

2.1 Related Work

The immigration process in FLOSS has been studied before. Prior works analyze the *attraction* of new immigrants to projects, *barriers* to their entry, and the *process* by which they join the project,

Several papers examine why FLOSS projects attract newcomers (see, among others *e.g.,* [12, 13, 14, 19])

. Suggested motivations include personal need for the software, reputation-seeking, and altruism. There are also inhibitors: programming is highly knowledge-intensive. Even experienced programmers have to work hard to gain specific skills needed for particular development tasks. This has been well reported [3, 5, 15]. A case study by Sim & Holt of immigrants in a *traditional* software project noted knowledge barriers to entry, and the importance of mentors [18]. 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 plenty of volunteers 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* study the immigration process of the Freenet FLOSS project [20]. They use data gathered from interviews, publicly available documents such as FAQs, email archives, and versioned source code repositories. They find 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 examines the Python project and the interactions of a particular individual as he transitions from a newcomer to a full-fledged developer [8]. He finds 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 [11]. 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 note the importance of these tools in keeping an OSS project organized.

In this paper, we also study immigration, but using *a quantitative approach*, based on hazard rate analysis. Hazard rate analysis, or survival analysis [6], is used to study time-dependent phenomena such as mortality, 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 use this method to study the immigration into FLOSS projects; *i.e.,* we model 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

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 three different mechanisms 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?

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.

Commitment 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 these people 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,and most likely to become developers(See von Krogh [20]).

The number of non-developers who remain active on the mailing list (in all the projects we studied) decays steadily as tenure increases to the maximum (*i.e, lifetime of the email archive*). In contrast, after much shorter tenure interval (relative to email archive lifetime) there are very few active *patch submitters* remaining. All three projects we considered show the same pattern (though with different time periods).

Knowledge and Skill Level can be expected to increase with the time a person spends with the project. 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 [5, 18]. 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. Getting their first patch accepted marks a milestone in skill acquisition. In the Postgres project, we find that the median time for first patch submission is during the second month of mailing list participation; median time for first patch acceptance is the third month of participation. For Apache, these median times are second and tenth month, and for Python, the median times are sixth month and thirteenth month respectively. These numbers indicate the time commitment required for skill acquisition.

Individual 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 [8, 20].

Social network theorists have developed validated network measures of community importance, based on the network of interactions [21]. 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 all the projects we studied we found that median in-degree, as a function of tenure, first increases, then flattens, 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.

SummaryAcceptance 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? 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 and/or acceptances, will increase the likelihood of becoming a developer.*

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

Note 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 [9] 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.

2.3 Project-Specific Considerations

Most mature FLOSS communities have policies that regulate how one gains write-access to the project repository, thus becoming a developer.

The Apache project is governed by the Project Management Committee (PMC) which makes decisions regarding major changes to the source code or documentation and grants write-access to developers through a voting system[1]. One can only become a member of the PMC through at least six months of contributing activity, nomination by an existing member, and unanimous approval by the current members. There is a much larger body of "committers" outside of this core group that we consider developers. Admission to

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

this group is less stringent, though still regulated and is the focus of our study.

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[2]. Acceptance into the latter category requires demonstration of technical expertise, but does not include as careful a screening process as the former.

The Python project does not appear to have a formal 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"[3] by Raymond Hettinger, a core Python developer. Tips include making ones self visible to the core developers through 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).

While these specific policies necessarily influence the developer immigration, we argue all of them are consistent with our conceptual framework: potential committers acquire skills, and display them on the mailing list through discussions and work gift-offerings, and will sometimes actually be given the right to commit. We argue therefore that our analysis still lends valuable insight into the determining factors of developerhood. Below are descriptions of the policies for each project studied.

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 [10, 16]. Extra effort was spent to ensure that email aliases and repository author identities were properly resolved, using automated and manual methods [1]. We built a social network from the email correspondence and computed social network measures [21] on a monthly and cumulative basis. We also extracted patch submissions from emails and searched the project repository for evidence of at least partial patch application[4]. Prior research has indicated the importance of patch submission and acceptance in gaining developer status [8]; 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 (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 binary variable *patch* indicates if this individual has previously submitted a patch. Second, as discussed earlier, we expect that *indegree*, 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 *success_pct* measures the percentage of patches submitted by this individual that were marked as accepted. 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

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

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

[4] We encourage the reader to see our companion submission to MSR 2007 located at `http://wwwcsif.cs.ucdavis.edu/~bird/papers/bird2007dps.pdf` for details of this process

server) only contained 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.

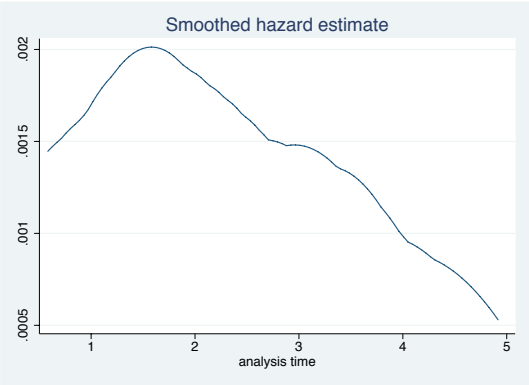

Figure 1. Fitted hazard rate estimate for immigration events in Postgres with time scale in years

After gathering the variables, we accounted for multicollinearity between variables by checking correlations. We omit these details here for brevity [5].

Variable	Mean	Std. Dev	Min	Max
(**Postgres**) n = 169,118 c = 3,283 i = 19				
patch	.04	.20	0	1
norm_indegree	.0024	.0065	0	.18
success_pct	1.27	9.78	0	100
devs_cum	13.92	4.14	1	20
sent_cum	13.12	63.55	1	2370
(**Apache**) n = 50,170 c = 1,445 i = 30				
patch	.10	.30	0	1
indegree	.0026	.0049	0	.087
success_pct	1.13	8.98	0	100
sent_cum	7.34	18.18	1	273
devs_cum	46.24	9.76	3	57
(**Python**) n = 45,216 c = 1,320 i = 62				
patch	.04	.20	0	1
norm_indegree	.0098	.032	0	.70
success_pct	.62	6.98	0	100
devs_cum	58.69	14.21	0	79
sent_cum	13.87	54.44	1	1648

Table 1. Univariate Statistics of the predictors

In figure 1, we show a smoothed plot of the the rate at which people become developers as a function of time. This curve suggests that the hazard rate does indeed increase and then decrease. This curve is estimated 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`

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

3.3 Hazard Rate Analysis: background

Hazard rate analysis, or *survival* analysis [2, 6], can model stochastic time-dependent phenomena such as cancer survival, employment duration etc. We use survival analysis to *understand* which factors influence, to what degree, the duration and occurrence of such events 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 [2] 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 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 acceptance 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 3 variables *patch*, *sent_cum*, and *success_pct*, which all have to do with an individual behavior and skill level. Finally, we added the *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 (3 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, patch, acceptance_rate	184	74	114
indegree	10	44	11.6

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 α_{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. 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 in the column (indicated with *), 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, *patch* is a binary variable that indicates previous patch submission history. The effect of this variable is *very* strong in Apache and Python: previous patch submit history in Apache increases the hazard rate by a factor $e^{2.71}$, about 15-fold. In Python, this effect is about $e^{2.09}$ or 8-fold. Previous *acceptance_rate* in getting patches accepted shows a strong, significant effect in Apache: the standard deviation of *acceptance_rate* in Apache is 8.98 (from Table 1). Thus the proportional increase in hazard rate one standard deviation change is $e^{8.98*0.15}$ or roughly 3.8-fold. The effect of this variable is not statistically significant in Postgres and Python. *Sent_cum* is the total number of messages sent by an individual. The effect in Apache is statistically significant, resulting in a $e^{0.014*63.55}$ (2.5 fold increase) in the rate for

Variable	Apache Model			Postgres Model			Python Model		
	Coef.	z	P >|Z|	Coef.	z	P >|Z|	Coef.	z	P >|Z|
tp1 (0-6 months)	-6.69	-2.97	0.003	-20.63	-10.62	0.000	1.10	0.67	0.5
tp2 (6-12)	-6.09	-2.66	0.008	-5.20	-2.32	0.021	1.18	0.69	0.492
tp3 (12-24)	-6.48	-2.7	0.007	-6.51	-2.9	0.004	0.098	0.06	0.955
tp4 (24-36)	-7.98	-2.93	0.003	-5.77	-3.00	0.003	-0.23	-0.13	0.9
tp5 (36-48)	-6.79	-2.53	0.011	-5.98	-2.88	0.004	0.10	0.06	0.96
tp6 (48-60)	-8.16	-2.60	0.009	-8.14	-2.88	0.004	0.91	0.47	0.64
indegree*	61.21	2.78	0.005	72.2	5.46	0.000	5.49	2.92	0.004
patch*	2.71	5.09	0.000	0.98	1.42	0.154	2.09	5.95	0.000
acceptance_rate*	0.15	2.03	0.043	0.0039	0.45	0.656	0.0062	0.10	0.918
sent_cum*	0.014	4.20	0.000	.00158	1.16	0.246	0.0041	1.86	0.062
devs_cum	0.14	1.84	0.066	1.09	2.17	0.03	0.14	2.81	0.005
time_trend	-0.91	-1.90	0.057	-2.06	-1.93	0.054	-1.46	-3.19	0.001

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 Apache, prior experience of *patch* acceptance (binary variable) increases the rate by $e^{2.71}$, or nearly 15 fold. *tp*1, *tp*2, *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 *tp*1 to *tp*4. Since there are very few non-developers that stay past 4 years, the interpretation beyond this point is unclear.

one standard deviation increase in value. There is no significant effect in other projects. The social network measure *indegree* is statistically very significant in all 3 models; however, the effect is more moderate, increasing the rate by about 34% in apache, 60% in Postgres and 19% 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, *tp*1 etc., constitute the pure time dependence of the hazard rate model. The proportional effect of the rate function can be interpreted as:

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

Each variable $tp1 \dots tp6$ should be interpreted as binary, taking on a value of 1 while t is 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 Python model does not have a good fit for the piecewise-constant time-dependent part of the hazard rate model.

Finally, we control for two potentially confounding variables: first, 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 $tp1 \dots$, which are tenure periods per individual. The model shows a negative effect, suggesting that all three projects become harder to join as time goes on. 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 appears to have a positive effect, perhaps indicating that in all three projects increasing developer pools makes it easier to gain admission.

Summary: The model results for the three projects are somewhat different; however, we can draw some conclusions from the fit of these models.

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 is possibly 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 Apache and Python, prior history of patch submission has a very strong effect. The effect is not statistically significant in Postgres. We found some difference between Postgres and the other projects that may account for this. In Postgres, a much greater proportion (50%) of patches are submitted by non-developers than in Apache (31%) and Python (23%). This would suggest that submitting patches is a behavior more common among non-developers in the Postgres project; therefore patch submitters are less distinguished, and have less of an advantage in gaining developer status. So we conclude: demonstrated skill level, via patch submission plays a very strong role in Apache and Python, but not so in Postgres. *Hypothesis 2 is supported in Apache and Python, but not in Postgres.*

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. So we conclude that *Hypothesis 3 is sup-*

ported in all three projects.

4 Threats to Validity

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

We inspect emails for patches, and then check if they were successfully applied. Although we are very confident in our ability to recognize submitted patches and acceptance of unmodified patches, detecting modified patches is difficult. A companion submission to MSR 2007 explains our approach to this problem.

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 [1, 8, 12, 16] 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[6].

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. While there is accepted literature [17] that supports the notion that contributing patch "work-gifts" is one the best ways to exhibit technical skill, there are other ways such as technical discussion that we do not capture.

There are limitations to how well these results may generalize to other FLOSS communities. For analysis, we needed projects with a long history of public archived data and a large community and developer base. These criteria necessarily introduce some bias into our results, which may not hold for other projects. Since we have the tools and theoretical infrastructure, we hope to test our hypotheses on other projects.

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 the Apache web server, Postgres, and Python projects, and used a piecewise-constant proportional hazard rate model to estimate these effects. Our three case studies lend support to these hypotheses.

[6]Please see `http://httpd.apache.org/ABOUT_APACHE.html#Development`

References

[1] 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.

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

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

[4] 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.

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

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