2013 IEEE International Conference on Software Maintenance

Social Activities Rival Patch Submission For Prediction of Developer Initiation in OSS Projects

Mohammad Gharehyazie
Computer Science Department
University of California, Davis
Davis, CA 95616
gharehyazie@ucdavis.edu

Daryl Posnett
Computer Science Department
University of California, Davis
Davis, CA 95616
dpposnett@ucdavis.edu

Vladimir Filkov
Computer Science Department
University of California, Davis
Davis, CA 95616
filkov@cs.ucdavis.edu

***Abstract*—Maintaining a productive and collaborative team of developers is essential to Open Source Software (OSS) success, and hinges upon the trust inherent among the team. Whether a project participant is initiated as a developer is a function of both his technical contributions and also his social interactions with other project participants. One's online social footprint is arguably easier to ascertain and gather than one's technical contributions *e.g.*, gathering patch submission information requires mining multiple sources with different formats, and then merging the aliases from these sources. In contrast to prior work, where patch submission was found to be an essential ingredient to achieving developer status, here we investigate the extent to which the likelihood of achieving that status can be modeled solely as a social network phenomenon. For 6 different OSS projects we compile and integrate a set of social measures of the communications network among OSS project participants and a set of technical measures, *i.e.* OSS developers patch submission activities. We use these sets to predict whether a project participant will become a developer. We find that the social network metrics, in particular the amount of two-way communication a person participates in, are more significant predictors of one's likelihood to becoming a developer. Further, we find that this is true to the extent that other predictors, *e.g.* patch submission info, need not be included in the models. In addition, we show that future developers are easy to identify with great fidelity when using the first three months of data of their social activities. Moreover, only the first month of their social links are a very useful predictor, coming within 10% of the three month data's predictions. Finally, we find that it is easier to become a developer earlier in the projects lifecycle than it is later as the project matures. These results should provide insight on the social nature of gaining trust and advancing in status in distributed projects.**

I. Introduction

Open Source Software (OSS) are developed by volunteers who are often geographically and temporally distributed, and yet work together effectively and productively. Well known examples of successful OSS projects, like the Linux operating system, Apache web server, and many others, rival or even exceed the quality of commercial competitors.

Participants in OSS projects have different roles, most prominently developers, patchers and users. Developers introduce changes to the code by *committing* directly to the project's *source code repository*; they have the highest level of access to the project and also share the greatest responsibility of delivering a viable product. Patchers submit proposed changes in the form of small updates to the code, known as *patches*, such as bug fixes, feature improvements, or documentation, which then are reviewed by developers and added to the project code at their discretion. Users are the downstream consumers of OSS. Project participants may migrate between these roles through the life of a project. This process has received a lot of attention in the empirical software research literature where it is variously referred to as developer initiation [1], entering the circle of trust [1], migration [2], and immigration [3]. A typical trajectory to becoming a developer is to start communicating with other participants in the project and then gradually get more involved, by earning a more central position in the project's social networks and/or producing more valuable technical contributions, like submitting patches, or working on bug activities [3].

The congruence of one's social and technical activities makes for a successful participation in a project [4]. In a sense, a project participant is as integral to the project as their contributions, be they communications or bug fixes. Strong working code leads to trust in the participant's ability to develop within the context of the project, and additionally, strong social skills signify to the team that the participant can be trusted to work within the project's team setting. The more trustworthy a participant, the more likely it is that he may eventually achieve developer status. Generally, only those participants who have sufficiently proven themselves through their activities, become developers.

Developer initiation, thus, depends on the social and technical actions of project participants, *e.g.*, who they talk to, the number of social links they have with other project participants, their communication patterns, patch submission activity, bug identification and fixing, *etc.*. But to what extent do social activities and technical activities work together to increase one's chance of advancing to the ranks of developers?

1063-6773/13 $26.00 © 2013 IEEE
DOI 10.1109/ICSM.2013.45

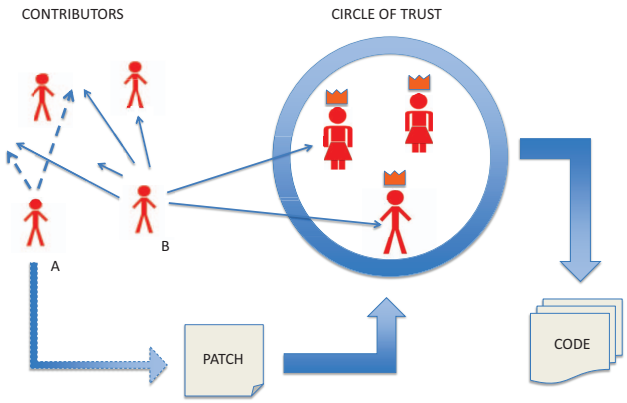

Figure 1. An OSS project community's circle of trust is comprised of developers. Project participants A and B are candidates for entering the circle. The activities of Person B are more significant positive predictors of him gaining developer status because he communicates more (thin solid arrows) than A does (thin dashed arrows), even though A contributes patches and B doesn't.

In this paper, we revisit the issue of migration between roles in OSS projects studying it from a social network analysis perspective. We collect developer communication, patch submissions, and code change data for 6 projects from the Apache Software Foundation (ASF). From the data we build statistical predictors for future developers in OSS projects and compare models with different predictors, as well as models across different time periods. We find that:

- Developer initiation in OSS can be modeled very well as a social network phenomenon based solely on people's communication activities, in particular the number of (two-way) social links they establish, *i.e.* messages participants *respond to* or *receive*, in response to their own message.
  The (two-way) social links are different from a one-way communication, most of which might not attract any attention or response. Social links based models exhibit better predictive ability for developer initiation than models incorporating patch submissions.
- Whether a participant will eventually become a developer can be predicted with great accuracy from the first three months of their tenure with the project. In most cases, this is based solely on the number of their (two-way) social links. Furthermore, models learned on only a single month of data containing the social links that one establishes, yields good predictions results that are within 10% of the three month data.
- The impact of both social and technical participation declines with project age. It becomes more difficult to attain developer status as the project matures.

The rest of the paper is organized as follows. We first focus on the background behind OSS development and migration and present our research questions. Then, we describe the data and data gathering process, followed by our methods, results, and conclusion sections.

A. Background

To become a new developer, a participant must first gain acceptance within the community. Trust in the developer's technical and social skills is believed to increase with time as a result of increased contribution and interaction by new participants [5]. Participation indicators, including the number of emails sent, their degree in the social network, the number of developers among a participant's neighbors, the numbers of bugs reported, and the number of patches submitted, may all be indicators of the community's trust in the candidate's abilities.

In most cases, to become a developer in an ASF project a new participant will first engage in discussion with other users and developers by joining the developer mailing list. Actively contributing interesting discussions and valuable insights to the project are key ways in which a new user can attract the attention of existing developers and gain social reputation within the community. The activity and the reputation of a user can be quantified through measures such as the number of emails he sends and the degree of the corresponding nodes in the email social network [6], [7].

Submitting patches is the preferred way in which unverified code changes are communicated in many OSS projects. Submitting a bug fix patch provides a basic degree of evidence that a user understands the software at a technical level. Such contributions should increase a participant's trustworthiness within the community. The contribution of the user can be quantified, *e.g.*, by the number of patches accepted.

In the ASF open source community, once a user has contributed sufficiently to a project, they may be nominated for developer status by an existing developer. The existing developers may then choose to grant this user committer status, allowing the new developer to make direct changes to the project's source code repository. Hence, whether a participant will eventually become a developer is a function of all the participants social and technical activities within the project's ecosystem.

Figure 1 illustrates two contributors, A and B, who want to become developers, but have two different profiles of activities. Contributor A contributes patches, and has a low level of communication with other participants, while B does not submit patches but communicates extensively with other participants. Our results in this paper show that B's activities are more significant for predicting his future developer status than are those of A, respectively.

It is important to note that this is just the basic framework of how a user becomes a developer in ASF projects, and the situations may vary for different projects in different stages of their evolution. For example, some projects may require the

users to enter the bugs into a bug tracking database, such as *Bugzilla* or *Jira*, while others may require users to submit bug reports to a mailing list. The latter method may increase the interaction between users and developers in these projects, and thus affect the dynamics of earning trust. Additionally, users may have different motivations to join a project [8]–[10], *e.g.*, enjoyment, reputation building, and skill improvement. The commercial backers of some open source projects may also provide incentives for skilled programmers to contribute in order to grow their project in it's early stages, while in the later stages, when the project has matured, developers are often more willing to volunteer in order to gain a signaling benefit to prospective employers [9].

B. Research Questions

In this paper we seek to identify effective social activity predictors of developer initiation in OSS projects, and to improve upon existing models for predicting future developers. Specifically, we ask which social metrics are effective, how do they interact with the technical measures of patch activities, and how soon can we predict that a participant will become a developer?

While the data on mailing list communications within projects are readily available, as are code commits and changes, patches and bug identification data is more difficult to gather [11] since it requires parsing message texts and mining multiple sources of predicting developers both when patch information is available and also when it is not.

> **Research Question 1:** To what extent can developer initiation in OSS projects be modeled as a function of patch activities and social communication? How about just as a function of social communications?

Early in their tenure as OSS project participants, people's patterns of technical and social activities are rapidly changing. Participants usually take some time to familiarize themselves with the code before submitting patch fixes. Additionally, they also might try to assimilate the project culture and available knowledge initially before asking questions of their own. Therefore, predictions of future status may be unreliable early in a person's tenure. Here, we seek to predict project participants' likelihood of becoming a developer based on the patterns of their activities early in their tenure.

> **Research Question 2:** How well can we predict if a person will become a developer based on information early in their tenure with the project? *i.e.* can we tell if someone will become a developer based on their activities in the first three months? Six months? Or just one month?

Finally, as projects evolve, determinants of trust are also likely to evolve and change, consequently, measures of trust and predictors of status change may not be static. This is mirrored in other fields, *e.g.* clandestine operations, where communication is over public channels but action traces are rarely readily observable. While the number of developers in a project grows proportional to its size, the number of participants in a project's mailing list grows exponentially. This increasing gap between potential developers and new position openings in turn change how trust is earned as a project matures.

> **Research Question 3:** Is it easier or more difficult to become a developer later in the project?

II. Related Work

There have been a fair number of studies on the motivations of developers for joining OSS projects and migration in OSS projects. Some projects have clear guidelines on how a new participant can contribute. The structure of this hierarchical process is known in the literature as the "onion model" [8], [12].

Von Krogh *et al.* performed a detailed case study of the freenet project; they interviewed participants and developers recording their patterns of individual activity and concluded that individuals following these guidelines are highly more likely to become developers [13]. Ducheneaut examined a single individual and his process of promotion to a core developer in the Python project [14]. Jensen and Scacchi studied role sets and the process of role migration in Mozilla, ASF and Netbeans [2].

Sinha *et al.* studied how developers enter the "circle of trust" by identifying key factors that lead to committer status [1]. They hypothesized that developers who contribute to the projects' bug tracking system, have prior experience contributing code to OSS, and who work for the same organization as some member of the core group, are more likely to obtain committer status.

The path to becoming a developer is not necessarily a step-by-step process. Herraiz *et al.* found that apart from gradual progression, there is another common developer joining pattern, *viz.*, the quick initiation of employees of enterprises invested in that OSS project as new developers [15]. Shibuya and Tamai performed case studies and confirmed these findings on other OSS projects (GNOME, OpenOffice.org, MySQL) [16].

Qureshi and Fang have identified different classes of developers based on socialization patterns using Growth Mixture models. They found that for each class of social behavior, the *"Lead Time" i.e.* the time it takes to become a developer, is unique and correlates with the amount of social activity of that class [17].

Bird *et al.* quantitatively modeled the relationship between time spent with the project and the probability of becoming a developer; their model used patch activities, social network attributes, and the time to first commit from the time of first

communication on an email network [3]. Using proportional hazard rate modeling they observed that a developer's *tenure* is related to his skill and commitment as measured by his participation in the email network and his contribution of patches prior to first commit. Further, they identified and described a non-monotonic trend in the likelihood of becoming a developer that rises with tenure, peaks, and then declines with project maturity. While their work is similar to ours in some aspects, their approach of predicting the "time" until one becomes a developer is in contrast to our work in that we focus on identifying "who" is more likely to become a developer rather than "when". The hazard analysis techniques used in their work support their approach.

Zhou and Mockus have modeled the status of "Long Term Contributors" based on three dimensions: environment, willingness, and capacity [18]. Their work focuses on issue tracking systems and workflows within those systems as sources of information.

Our work differs in that we focus on understanding how soon can we predict developer initiation after initial participation and, additionally, to what degree can social metrics replace technical attributes.

III. Data Gathering

In this study we focus on six projects from the Apache Software Foundation. Their summary is given in Table I. For each project we mined three datasets, the *developer* mailing list, the issue/bug tracking systems and the source code repository.

From the developer mailing lists, we construct the Email Social Networks, *ESN*. We extract patch submissions from both the issue tracking systems and the developer mailing lists. Lastly, we obtain the history of source code changes from the source code repository. All of the sources were mined from the first date of available data, until the date of mining (March 2012), and the dates in Table I denote the intersection of available data in all three datasets.

We extract patch submissions from both the issue tracking systems and the developer mailing lists. Lastly, we obtain the history of source code changes from the source code repository.

TABLE I. OSS projects used in this study show diversity both in size and in relative activity. #Users refers to the number of individuals in the ESN and #Devs refers to the number of distinct developers in each project's source code repository.

Project	#Users	#Devs	#Mails	#Patches	Start	End
Ant	1416	44	17300	1482	2000-01	2012-03
Axis2_c	600	24	11152	754	2004-01	2012-03
Log4j	539	18	3811	166	2000-12	2012-03
Lucene	2155	41	43922	5576	2001-09	2012-02
Pluto	266	24	3017	259	2003-10	2011-09
Solr	840	19	14411	4090	2006-01	2010-04

A. Email Social Networks

It is a general policy for OSS projects to channel as much communication as possible through project mailing lists so that all participants can benefit from the exchange of ideas and information [11]. Any message sent to these lists will be broadcast to all subscribed participants. Such broadcast messages differ from point to point email messages in that they do not have a clear and distinct recipient. Even given this difficulty, however, we can still infer a communication network. When person A sends message M, in response to message N, that was sent by person B, then there is a high chance that A primarily intended to communicate to B in response to their initial message that was broadcast to no particular recipient [11]. Such links are in fact two-way social links, in that the communication occurs both ways. Self-loops were removed when constructing ESNs. We note that the final network is a *multigraph*, since we permit multiple edges between people.

1) Unmasking Aliases: Project participants often use different aliases within the same project, *e.g.*, (John Smith, smith@gmail.com), (Smith, John@smith.com), (John S., J.smith@ucdavis.edu). Since these aliases represent a single person they must be merged if we are to accurately capture an individual's social activity. Unmasking aliases is a well-known problem in the literature [11], [19], [20]. Bird *et al.* used a string similarity based approach to address this problem [11]. We improved and automated their procedure reducing the need for human interaction. We first remove all suffixes, prefixes, and generic names, *e.g.*, Dr., Mr., Jr., Admin, for comparison. Then, for each alias pair a similarity score is calculated as a combination of four sub-scores comparing aliases' names and emails. Perfect matches are automatically merged, whereas less than perfect matches that achieve a minimum threshold of 0.93 on a unit scale are presented to the researcher to disambiguate. The chosen threshold was selected empirically to reduce the number of false positives while limiting false negatives.

B. Issue Tracking Systems

Issue tracking systems such as Jira and Bugzilla maintain a database of issue reports submitted by developers and users. Issues are of various types including new features, improvements, or defects. Developers can submit new issues to the tracking system or report results on existing issues by interacting with the system either through the web, or in some cases, directly via the source code repository.

C. Patch Extraction

Patches are the preferred method of fixing bugs or issues and/or making small adjustments to the code. Patch submission is not limited to developers, and submitting a patch does not imply that the mentioned patch will be applied. It is up to developers to review the patches and apply them if they are accepted.

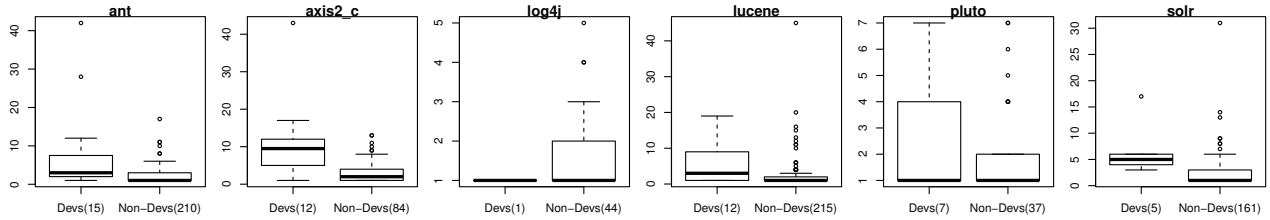

Figure 2. Distribution of number of patch submissions by developers and non-developers in each project. Only individuals with at least one patch submission are plotted, since adding those with no submission would highly skew the plots towards zero. The numbers in the parentheses show each group's population. A Wilcox signed rank test across each project yields p-values in order: 0, 0.01, 0.44, 0, 0.72, 0, indicating that in Log4j and Pluto, patch submission is not statistically different for those participants who become developers and those who don't when measured in the first three months of participation.

There are multiple methods and formats in which one can submit a patch to an OSS project. The common accepted format for a patch submission is to send the "diff" of the code changes to existing developers. ASF provides two popular choices of issue tracking systems for OSS projects: Jira and Bugzilla. Patches can be submitted as attachments to an issue, or simply as inline text, *i.e.* pasting the patch "diff" code, either as part of the issue's description text, or as a comment on that specific issue.

The developer mailing lists can also act as a medium for patch submission [21]. In this case patches are submitted through messages to the mailing list, either with the code pasted in the text message or as an attachment.

To extract inline patches, we used regular expression pattern matching queries in mailing list messages, and issues/bugs comments and description. Patches submitted as attachments are generally named with a ".diff" or ".patch" extension. However, we found that sometimes patch submissions do not follow this naming convention, e.g. "patch.txt" is used, or multiple patch files are combined in an archive, requiring the extraction of that file, and subsequent manual examination of all filenames contained within the archive.

Table II shows the distribution of patch submission among the multiple sources of patches in each project.

D. Source Code Repository

A source code repository and version control system, *e.g.* Git, SVN, and CVS, facilitates collaboration among developers by maintaining a history of changes and an associated log entry for each change. These systems can provide information such as list of files, list of developers and a detailed record of all changes to all files by any developer. This information allows us to distinguish developers from non-developers in the ESNs and to track their history of activity.

All selected OSS projects for this study use Git as their version control system, however they initially used SVN migrating later to Git, Some projects continue to maintain both repositories.

Since this dataset and the ESN use separate sets of aliases (not necessarily identical) as personal identifiers, a manual matching between people across the two datasets was performed.

TABLE II. Different projects have different paths for patch submission. The preferred method of patch submission is differs across projects and we attribute this to project culture.

	ML_inline	ML_attach	bugzilla_attach	jira_inline	jira_attach
Ant	413	976	93	0	0
Axis2_c	63	200	0	27	464
Log4j	87	57	19	0	3
Lucene	141	107	0	88	5240
Pluto	15	26	0	8	210
Solr	30	4	0	48	4008

IV. Methodology and Modeling

A. Social Communication Measures

For each participant in our datasets, we gather the following metrics to model the relationship between their social and technical characteristics to their potential developer status.

- *Project age* The number of days from the start of the project where a node has its first edge in the graph. This is the first message received by or replied by that person in the ESN.
- *Is a New Developer:* A binary variable indicating whether this person ever commits into the repository *after* they have joined the mailing list. Those who were developers prior to this time are beyond the scope of this paper as we are interested in the process of attaining developer status.
- *Number of Patches:* The number of patches one has submitted.
- *Number of Messages:* The number of edges connected to a node in the multigraph *i.e.* a node's degree. This is not just the number of messages one sends, but rather the number of messages one **sends** *in response to* others plus the number of messages one **receives** *in response to* their messages.
- *Number of Threads:* The number of threads started by a person. A thread is a message that is *not* in response to other messages. Threads are not included in the ESN due to inability to associate them to a specific pair of nodes.

TABLE III. Patch submission is a significant predictor when no social variables are included in the model. This basic logistic regression model only uses "number of patches" in 3 months and includes "project age" as a control variable. For all variables, the log of that variable plus 0.5 was used in the modeling. The values in the first column are the model coefficients and the highlighted coefficients are statistically significant ($p < 0.05$).

Ant	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	-1.73	1.12	-1.55	0.12
Project age	-0.33	0.18	-1.87	0.06
Number of patches	**1.06**	0.18	5.82	0
Axis2_c	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	-0.91	1.01	-0.9	0.37
Project age	**-0.39**	0.16	-2.46	0.01
Number of patches	**0.93**	0.21	4.53	0
Log4j	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	-3.53	1.98	-1.78	0.07
Project age	-0.11	0.29	-0.38	0.7
Number of patches	0.36	0.83	0.43	0.67
Lucene	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	1.08	0.84	1.28	0.2
Project age	**-0.7**	0.12	-5.71	0
Number of patches	**1.16**	0.18	6.42	0
Pluto	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	-0.88	0.84	-1.05	0.3
Project age	**-0.34**	0.15	-2.3	0.02
Number of patches	**1.11**	0.32	3.45	0
Solr	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	0.44	1.19	0.37	0.71
Project age	**-0.72**	0.19	-3.69	0
Number of patches	**0.75**	0.29	2.58	0.01

TABLE IV. The second logistic regression model, adding "number of messages" to the previous model. It is seen that "number of patches" slightly loses its significance.

Ant	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	**-4.32**	1.32	-3.28	0
Project age	-0.2	0.19	-1.08	0.28
Number of patches	**0.61**	0.2	3.06	0
Number of messages	**1.07**	0.2	5.35	0
Axis2_c	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	**-2.85**	1.33	-2.15	0.03
Project age	-0.3	0.17	-1.81	0.07
Number of patches	**0.53**	0.25	2.11	0.03
Number of messages	**0.57**	0.22	2.62	0.01
Log4j	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	**-7.28**	2.51	-2.9	0
Project age	-0.13	0.3	-0.42	0.67
Number of patches	-0.8	0.98	-0.82	0.41
Number of messages	**1.89**	0.44	4.26	0
Lucene	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	0.26	0.89	0.29	0.77
Project age	**-0.8**	0.13	-6.04	0
Number of patches	**0.69**	0.22	3.14	0
Number of messages	**0.74**	0.2	3.75	0
Pluto	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	-1.44	1	-1.44	0.15
Project age	**-0.37**	0.15	-2.46	0.01
Number of patches	0.81	0.42	1.95	0.05
Number of messages	0.42	0.4	1.04	0.3
Solr	Estimate	Std. Error	z value	Pr(> |z|)
(Intercept)	**-2.86**	1.45	-1.97	0.05
Project age	**-0.67**	0.2	-3.39	0
Number of patches	-0.15	0.35	-0.44	0.66
Number of messages	**1.18**	0.31	3.83	0

- *Neighbors:* The number of unique nodes that a node is connected to. This is different from number of messages in that a node can connect to other nodes through multiple edges thus differing these two measurements.
- *Neighbor Developers:* The number of unique nodes with developer status that a given node is connected to.

B. Modeling Developer Initiation

We use logistic regression, a generalized linear model designed to model probabilities for dichotomous outcomes, to model whether or not a project participant will become a developer based on several social explanatory variables.

Previous work in this area has used survival modeling to model the trajectory over time of developer initiation [3]. In this work we are focused on early detection of developer initiation which limits the amount of data available to model the trajectory. Moreover, since we are interested in the dichotomous outcome of whether a participant becomes a developer, logistic regression is a more appropriate choice.

The dataset used for our studies, contains the features and metrics described above for the first k months, of each individual's activity ($k = 3$ unless explicitly stated). We attempt to predict *"Is a New Developer"*. The variables used for each of the models are described in the results. The project age at which one joins that project, is added to all models as a control variable. For all numeric variables, the log of that variable plus 0.5 was used to stabilize variance and reduce heteroscedasticity [22]. Since all untransformed values in our data are skewed, the increase in the value of a variable by one unit does not have the same effect at high values as it does in low values, *e.g.*, the number of patches changing from 1 to 2 is much more meaningful than changing from 100 to 101.

Since we are trying to predict who *is going to* become a developer, sample data for developers who were initiated in fewer than k months were removed from the dataset.

For each learned model, we evaluate its validity based on two criteria. The first is project independence, *i.e.* we want our model to hold across projects. In support of this, we look at the stability of each model coefficient's statistical significance, as determined by the coefficient's *p-value*. Commonly, a p-value of less than 0.05 is an indicator of significant results.

Excessive multicollinearity is a concern in regression models and it can occur when predictors are highly correlated. To check for this we use the *Variance Inflation Factor (VIF)*. A common rule of thumb is that for any variable x in a model, $VIF(x) \geqslant 5$ indicates high collinearity. In all of our models in this paper, VIF of all variables remained well below 2 except for some models with highly correlated variables. These models were discarded as it will be explained later in the paper.

To evaluate a model's predictive power, we use the *Area Under the Receiver Operating Characteristic (AUROC)* measure [22]. ROC illustrates the performance of a binary classi-

fier in a TP-FP space, while varying the cutoff threshold. A random predictor would be a line with the slope of 1 and the area of 0.5 while a perfect predictor will have an area of 1.

Overfitting is a concern with any statistical model so to help alleviate any concern and to yield a stable estimate of the predictive power of our models we employ resampling methods. We define training and testing sets using $2/3$ holdout for our training sets. To maintain a similar distribution of developers vs. non developers in the training and testing sets we employ stratified sampling. Each of the test and training sets will then have roughly the same ratio of developers to non-developers. This ensures that the resulting model is not extremely biased in that the training set would contain almost all, or none of the developers. The first case would cause a testing set with no positive samples, and the latter would results in a zero model due to the lack of positive samples in the training set. We resample 250 times and average the AUROC over all these models to indicate the overall predictive power of a model.

V. Results and Discussion

A. Research Question 1

We evaluate here the stability and predictive power of models using patches, length of time with the project, and a number of social measures. We motivate this question with Fig. 2. This figure shows that, although most of the time there is a meaningful difference between developers and non-developers who submit patches, still there are many non-developers who behave like developers in terms of patch submission. Consequently, it is likely that using patches alone may not yield the best prediction model.

This simple model, using "Number of patches" and no social network measures shows that patch submission is often a statistically significant predictor (Table III). However, adding "number of messages" (as an indicator of social collaboration) to this model results in patches slightly losing their significance (Table IV), We used a Chi-Squared goodness of fit statistic to verify that the additional predictor explained a statistically significant amount of the deviance in the model, *viz.*, is the addition of the new variable justified. For all projects projects except Pluto adding "number of messages" was significant with a Chi-Squared test p-value of < 0.01. A Spearman correlation test between "number of messages" and "number of patches" does not show significant correlation between them, mostly below 0.3 over all projects, in concordance with our VIF values of less than 2.

The predictive power of these two models and the additional models with only "Number of messages" and the combination "Number of messages + Threads" is shown in Fig. 3. We see that not only adding number of messages dramatically improves the predictive power, but removing the patches variable from the model does not lower the predictive power of the model. A Kruskal-Wallace test followed by a post-hoc pairwise Wilcoxon test for each project reveals that in all projects except Pluto and Lucene either the models with messages or messages and threads have the highest mean AUC, and this difference is statistically significant. In Lucene, the best model uses both patches and messages. In Pluto, although the patches model has the highest AUC, there is no statistical difference between the models. Using patch information alone is not a bad predictor, but it is evident that using social network metrics yields more accurate predictions.

TABLE V. Spearman's Correlation between different variables in all projects. Correlation values higher than 0.5 are highlighted.

	Ant	Axis2_c	Log4j	Lucene	Pluto	Solr
messages vs. dev neighbors	**0.62**	**0.52**	0.49	**0.55**	**0.53**	**0.65**
neighbors vs. dev neighbors	**0.69**	**0.61**	**0.60**	**0.72**	**0.72**	**0.80**
messages vs. neighbors	**0.82**	**0.76**	**0.70**	**0.67**	**0.66**	**0.75**
threads vs. dev neighbors	0.21	0.38	0.12	0.38	0.25	**0.56**
threads vs. neighbors	0.31	0.54	0.14	0.37	0.25	0.48
threads vs. messages	0.31	**0.74**	0.17	**0.61**	**0.53**	**0.64**

We ask next whether we can improve this simple model by adding additional features from the ESN. Since we have observed that sending and receiving messages is an important indicator of whether someone will become a developer or not, naturally we ask whether it is the number of messages that is important or the number of distinct individuals one keeps in contact with? More precisely, are these contacts the same, or is communicating with developers more important than communicating with other participants? Also we want to see whether starting threads and discussions in contrast to replying and being replied to, is also an important factor in gaining the trust of the community.

These variables are quite highly correlated and we expect that this will impact model performance (The spearman correlation between these variables can be seen in Table V). We added the number of started threads, neighbors and neighboring developers to our existing models. While some predictors are statistically significant in some models as can be seen in Table VII, most are hampered by high variance

TABLE VI. Social metrics yield better performing predictive models for developer status across most projects. Mean AUC values over 250 runs using stratified sampling for each project. Italicized values indicate models that include an insignificant variable in the explanatory model. Values in bold are the highest mean AUC value over all models that remained significant after a post-hoc pairwise Wilcox test out of all explanatory models with significant variables. Projects that do not have a value in bold were statistically indistinguishable.

project	patches	patches+msgs	msgs	msgs+threads
Ant	0.71	0.87	0.87	**0.89**
Axis2_c	0.83	0.84	0.83	*0.82*
Log4j	0.30	0.75	**0.91**	*0.88*
Lucene	0.84	**0.85**	0.83	*0.84*
Pluto	0.79	*0.77*	0.76	*0.74*
Solr	0.76	0.90	0.91	**0.96**

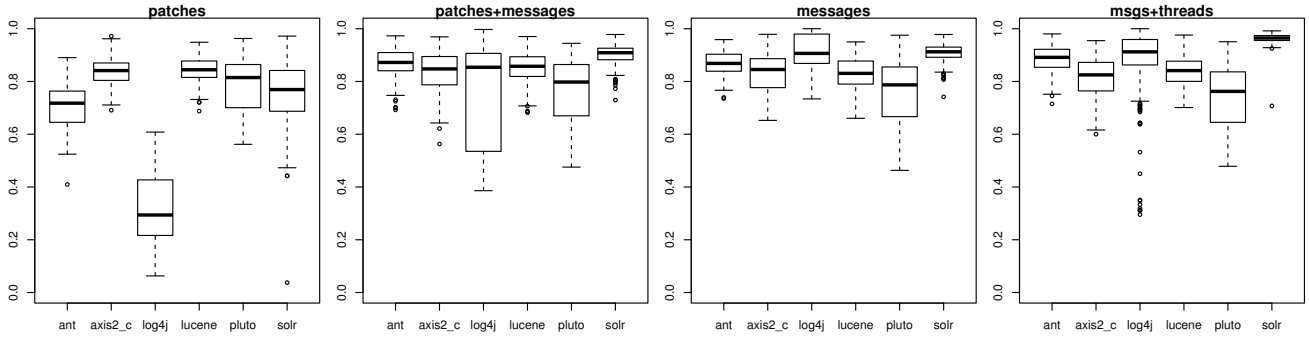

Figure 3. Social measures outperform patch submission in a predictive setting. AUROC of 4 models, on 250 iterations of modeling using stratified data.

TABLE VII. High multicollinearity limits the effectiveness of additional social variables. None of the added social network measures are stable across projects. Number of threads is significant in two projects, ant, and solr.

	Ant	Axis2_c	Log4j	Lucene	Pluto	Solr
(Intercept)	**-4.13**	**-4.24**	**-6.28**	-0.27	**-2.49**	-2.78
Number of messages	**1.2**	**0.9**	**1.67**	**1**	0.58	**1.01**
Number of neighbor devs	0.08	-0.26	0.25	0.05	0.72	0.27
Project age	-0.29	-0.21	-0.17	**-0.83**	**-0.33**	**-0.65**
(Intercept)	**-5.64**	**-3.78**	**-6.87**	-0.39	**-2.04**	**-5.33**
Number of messages	**1.04**	**0.82**	**1.51**	**1.14**	**0.69**	**2.74**
Number of threads	**0.69**	-0.01	0.6	-0.15	0.28	**-1.26**
Project age	-0.17	-0.26	-0.12	**-0.82**	**-0.43**	**-0.77**
(Intercept)	**-5.7**	**-4.71**	**-16.87**	-1.66	**-2.66**	**-5.64**
Number of messages	**1.01**	0.21	-2.09	0.25	0.23	**2.21**
Number of threads	**0.69**	0.12	**1.07**	-0.03	0.34	**-1.24**
Number of neighbors	0.05	1.12	**7.3**	**1.58**	0.89	1.08
Project age	-0.16	-0.15	0.9	**-0.67**	**-0.34**	**-0.74**
(Intercept)	**-5.63**	**-4.28**	**-7.06**	-0.39	**-2.26**	**-5.56**
Number of messages	**1.03**	**0.94**	1.15	**1.12**	0.34	**2.6**
Number of threads	**0.69**	-0.04	0.68	-0.14	0.35	**-1.27**
Number of neighbor devs	0.02	-0.27	1.02	0.03	0.85	0.37
Project age	-0.17	-0.21	-0.11	**-0.82**	**-0.39**	**-0.72**

inflation factor owing to the high correlation between "Number of Messages", "Number of neighbors", and "Number of developers" Table V. "Number of threads", however, was significant in two projects, Ant, and Solr, and had sufficiently low variance inflation factor that inclusion of the variable improved prediction results. We discuss this further in the next subsection.

Result 1: *Developer initiation can be modeled using social activity alone, performing no worse than models which also incorporate patch submission. The basic model of social activity only uses "Number of Messages", however adding "Number of Threads" improved prediction results in 2 of the projects, hinting this might be a matter of "project culture".*

TABLE VIII. Number of messages is a statistically significant predictor with as little as only one month of data. Stability of models with log of number of messages, for 1 to 6 months. Models using two or three months time window are slightly more stable across all projects

	Ant	Axis2_c	Log4j	Lucene	Pluto	Solr
(Intercept)	**-3.04**	**-2.97**	**-2.94**	0.55	-1.43	0.42
Messages in 1 month	**1.09**	**0.73**	**1.43**	**0.72**	0.49	0.48
Project age	**-0.34**	-0.27	**-0.45**	**-0.81**	**-0.36**	**-0.84**
(Intercept)	**-3.63**	**-3.64**	**-5.8**	-0.09	**-1.8**	-0.92
Messages in 2 months	**1.2**	**0.83**	**1.63**	**0.87**	0.46	**0.83**
Project age	-0.32	-0.24	-0.17	**-0.79**	**-0.31**	**-0.78**
(Intercept)	**-4.15**	**-3.77**	**-6.3**	-0.25	**-2.24**	-2.64
Messages in 3 months	**1.24**	**0.81**	**1.76**	**1.02**	**0.84**	**1.11**
Project age	-0.29	-0.26	-0.17	**-0.83**	**-0.38**	**-0.67**
(Intercept)	**-4.9**	**-3.27**	**-6.75**	-0.83	**-3.24**	**-3.13**
Messages in 4 months	**1.4**	**0.68**	**1.77**	**1.13**	**0.91**	**1.31**
Project age	-0.24	-0.32	-0.15	**-0.81**	-0.27	**-0.72**
(Intercept)	**-6.05**	**-3.51**	**-7.74**	-0.92	**-4.15**	**-3.37**
Messages in 5 months	**1.42**	**0.7**	**1.86**	**1.13**	**0.96**	**1.32**
Project age	-0.1	-0.32	-0.1	**-0.82**	-0.18	**-0.72**
(Intercept)	**-6.73**	**-3.48**	**-7.78**	-0.98	**-4.46**	**-3.57**
Messages in 6 months	**1.4**	**0.69**	**1.8**	**1.1**	**1.07**	**1.41**
Project age	0	-0.34	-0.08	**-0.82**	-0.19	**-0.77**

B. Research Question 2

In the previous section we used information on the first three months of individuals' activity to model their likelihood of obtaining committer status. But how early can such models provide useful predictions? Is one month of information sufficient, or should we increase to 6 months or more to yield better prediction models?

A limitation in evaluating models for longer time periods is that participants who become developers in a shorter period must be discarded from the training set, yielding a smaller dataset and consequently a less reliable model. The median time to become a developer in our OSS projects ranges from 8 months to almost a year (except for Log4j which is 4 months), and a range from 1 to 6 months includes more than half of the developers in the dataset.

To evaluate the sufficient-time-for-prediction hypothesis, we use the simple model discussed in the previous section (only

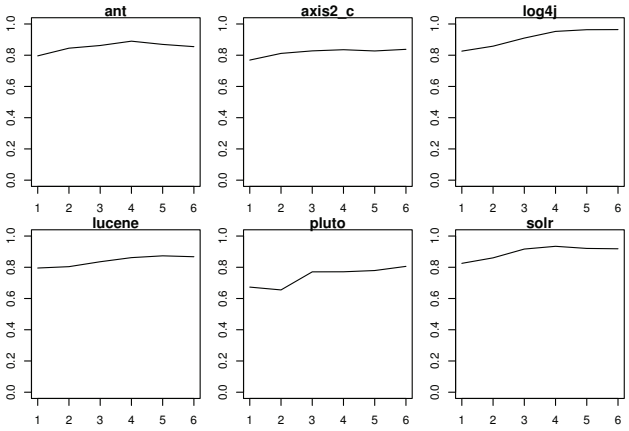

Figure 4. The predictive power of the model using "number of messages" from 1 to 6 months, each on 250 iterations of modeling using stratified data. The AUROC for each project slightly improves until 3rd or 4th month, and then stabilizes.

using "Number of Messages" as a predictor) with information on the first $n = 1, 2, ..., 6$ months of each participant's activity in the OSS' ESN. The results can be seen in Table VIII. For one or two months the models are not as significant as other models. But afterwards all the models are statistically significant, valid, and surprisingly stable.

Model stability only tells one part of the story, *viz.*, it can explain how the model fits the data. However, there is always risk of over-training and evaluation of the predictive power of the models will more effectively demonstrate the value of this model in a realistic setting. We see in Fig. 4 that the predictive powers of the models differ slightly from one time window to another. Time windows less than 3 months slightly suffer from low predictive power and time windows of greater than 4 months are almost no better than 3 or 4 months. We choose 3 months as our default because of best overall stability and predictive power (4 months is almost just as good, but with our goal of prediction, the smaller the time window, the better).

Additionally, adding the number of threads to our model improved the prediction results in two projects. Fig. 3 (bottom) and Fig. 5 show that adding the number of threads to the model slightly improves prediction performance.

Result 2: *Developer initiation can be modeled with as little as one month's information about the social activity of individuals; using three months yields stronger and more stable result.*

C. Research Question 3

We are also interested in understanding how trust evolves over the life of each project. Looking back at previous models, we see that in all cases, the coefficient for "Project Age" is negative, implying it becomes increasingly difficult to become a developer over time. To verify this hypothesis, we

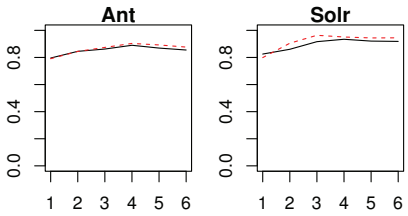

Figure 5. The AUROC of two projects with two different models. Black lines represent models using only "Number of messages" while red dashed lines represent models with "Number of Threads" added to them. The latter performs slightly better than the former.

replaced "Project Age" with a dummy binary variable called "IsSecond" which is true for the second half of population (sorted in ascending order by their "Project Age") and is FALSE for the first half. If the coefficient of this variable is still negative across projects it will confirm our hypothesis that being initiated a developer becomes increasingly more difficult over time. Ideally "Project Age" should be broken to smaller partitions (4 or more) to give us a higher resolution view, but increasing the resolution would result in even less sample points in each partition, making the results less reliable.

Two different logistic regression models were fit to the data, and the models are given in Table IX. We see that for all projects, the coefficient of "IsSecond" is negative, although only statistically significant across three of the projects. Based on these observations, we conclude:

Result 3: *It becomes more difficult for individuals to become developers as the project matures; late stage developers may have to put more effort to gain the same level of trust.*

VI. Conclusion and Threats to Validity

We presented strong evidence for the determining role that social networking activities play in becoming a developer in an OSS project. Surprisingly, to this end, social communications are a better predictor than patching activity. We also present

TABLE IX. It becomes increasingly more difficult to earn trust in an OSS. Models show a dummy variable "IsSecond" which is true for individuals that $p\_age > median(p\_age)$. It is seen that joining the project later has a negative effect on one's chance of becoming a developer.

	Ant	Axis2_c	Log4j	Lucene	Pluto	Solr
(Intercept)	**-5.5**	**-4.21**	**-7.69**	**-4.74**	**-2.99**	**-6.59**
Number of patches	**0.62**	**0.59**	-0.76	**0.62**	**0.85**	-0.2
Number of messages	**1.08**	**0.56**	**1.89**	**0.72**	0.49	**1.17**
IsSecond	-0.36	**-2.08**	-0.9	**-2.46**	**-2.1**	-1.34
(Intercept)	**-5.76**	**-4.93**	**-7.04**	**-5.42**	**-3.8**	**-6.33**
Number of messages	**1.24**	**0.82**	**1.78**	**0.99**	**0.88**	**1.07**
IsSecond	-0.57	**-1.84**	-0.99	**-2.67**	**-2.01**	-1.29

evidence that developers' early social activities in the project identify them as such. Expectedly, we also find that community trust is more difficult to attain with time as the community likely takes longer to identify trustworthy contributors.

Our methods are based solely on (two-way) social links representing messages sent between project participants, but is oblivious to the content of those messages. Clearly, knowing the content of the emails would add another layer of information that can be mined. However, the quality of our predictions while disregarding content is an indication of the strong influence of the social link structure. This may be of independent interest to the management and security communities.

Our results in no way imply causality, rather a strong statistical correlation between the measured attributes that can be used for prediction and further research.

We recognize several threats to the validity of our approach and conclusions. The dataset gathered here was from 6 projects, all from the Apache Software Foundation. This might impose a limitation on the pattern of communication and contribution in these projects that will limit the applicability of our results to other OSS projects. It also may be that there is a systematic bias in our data, meaning what we measure is not the likelihood of obtaining developer status *e.g.* people may be assigned to be developers (rather than being chosen) and are using ESNs to familiarize themselves with the community. Although this assumption is quite contrary to ASF's guidelines [23], it is not hard to imagine other scenarios where developers are not chosen as we think they are.

Having more projects is desirable, but practically, we had to select projects with a large number of developers for the predictive models to have reasonable statistical power. We cannot address private communication between developers which may impact the structure of the social network. This limitation, however, affects all such work of this nature and we do not believe that it severely limits the usefulness of our results.

VII. Acknowledgements

The authors would like to thank Bogdan Vasilescu for his contributions in mining issue tracking systems and mailing lists attachments, and for his insightful comments and feedback on this work. All authors gratefully acknowledge support from the Air Force Office of Scientific Research, award FA955-11-1-0246.

References

[1] V. Sinha, S. Mani, and S. Sinha, "Entering the circle of trust: developer initiation as committers in open-source projects," in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 133–142.

[2] C. Jensen and W. Scacchi, "Role migration and advancement processes in ossd projects: A comparative case study," in *Software Engineering, 2007. ICSE 2007. 29th International Conference on*. IEEE, 2007, pp. 364–374.

[3] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu, "Open borders? immigration in open source projects," in *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*. IEEE, 2007, pp. 6–6.

[4] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 2–11.

[5] K. Stewart and S. Gosain, "An exploratory study of ideology and trust in open source development groups," in *Proceedings of the 22nd International Conference on Information Systems*. ACM, 2001, pp. 1–6.

[6] M. Newman, S. Forrest, and J. Balthrop, "Email networks and the spread of computer viruses," *Physical Review E*, vol. 66, no. 3, pp. 035 101(R):1–4, 2002.

[7] C. Fershtman and N. Gandal, "Direct and indirect knowledge spillovers: the "social network" of open-source projects," *The RAND Journal of Economics*, vol. 42, no. 1, pp. 70–91, 2011.

[8] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," in *Proceedings of the 25th International Conference on Software Engineering*. IEEE, 2003, pp. 419–429.

[9] G. Krogh and E. Hippel, "The promise of research on open source software," *Management Science*, vol. 52, no. 7, pp. 975–983, 2006.

[10] W. Scacchi, "Free/open source software development: Recent research results and methods," *Advances in Computers*, vol. 69, pp. 243–295, 2007.

[11] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 137–143.

[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*, vol. 32, no. 7, pp. 1159 – 1177, 2003, open Source Software Development. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0048733303000477

[13] G. Von Krogh, S. Spaeth, and K. Lakhani, "Community, joining, and specialization in open source software innovation: a case study," *Research Policy*, vol. 32, no. 7, pp. 1217–1241, 2003.

[14] N. Ducheneaut, "Socialization in an open source software community: A socio-technical analysis," *Computer Supported Cooperative Work (CSCW)*, vol. 14, no. 4, pp. 323–368, 2005.

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

[16] B. Shibuya and T. Tamai, "Understanding the process of participating in open source communities," in *Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS'09. ICSE Workshop on*. IEEE, 2009, pp. 1–6.

[17] I. Qureshi and Y. Fang, "Socialization in open source software projects: A growth mixture modeling approach," *Organizational Research Methods*, vol. 14, no. 1, pp. 208–238, 2011.

[18] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in oss community," in *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 2012, pp. 518–528.

[19] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens, "On the variation and specialisation of workloada case study of the gnome ecosystem community," *Empirical Software Engineering*, pp. 1–54, 2013.

[20] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "Who's who in gnome: Using lsa to merge software repository identities," in *ICSM*. IEEE Computer Society, 2012, pp. 592–595.

[21] C. Bird, A. Gourley, and P. Devanbu, "Detecting patch submission and acceptance in oss projects," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*. IEEE Computer Society, 2007, p. 26.

[22] J. Cohen, *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.

[23] (2013) Apache software foundation - frequently asked questions. [Online]. Available: http://www.apache.org/foundation/faq.html