Noname manuscript No.
(will be inserted by the editor)

Developer Initiation and Social Interactions in OSS: A Case Study of the Apache Software Foundation

Mohammad Gharehyazie · Daryl Posnett · Bogdan Vasilescu · Vladimir Filkov

the date of receipt and acceptance should be inserted later

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 committer 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 committer 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 Apache Software Foundation 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 committer, and to characterize their socialization patterns around the time of becoming committer. 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 committer. 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 committers 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%

M. Gharehyazie, D. Posnett and V. Filkov
Computer Science Department
University of California, Davis
Davis, CA, 95616
E-mail: gharehyazie@ucdavis.edu, dpposnett@ucdavis.edu, filkov@cs.ucdavis.edu

B. Vasilescu
Department of Mathematics and Computer Science
Eindhoven University of Technology
5600 MB Eindhoven, The Netherlands
E-mail: b.n.vasilescu@tue.nl

of the three month data's predictions. Interestingly, we find that on average, for each project, one's level of socialization ramps up before the time of becoming a committer. After obtaining committer status, their social behavior is more individualized, falling into few distinct modes of behavior. In a significant number of projects, immediately after the initiation there is a notable social cooling-off period. Finally, we find that it is easier to become a developer earlier in the projects life cycle 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.

Keywords Open Source Software, Email Social Networks, Logistic Regression, Developer Initiation

1 Introduction

Open Source Software (OSS) are developed by communities of geographically- and temporally-distributed contributors ranging from professional software developers to volunteers from varied backgrounds who, despite participating in a very decentralized process, succeed to work together effectively and productively [1, 2].

Well known examples of thriving OSS projects, like the Linux operating system, Apache web server, and many others, rival or even exceed the quality of commercial competitors [3].

Although typically lacking the organizational hierarchy characteristic of commercial settings, most OSS communities create and enforce clear cut contribution policies. The resulting community structure, commonly referred to as the "onion model" [4–6], comprises different contributor roles based on their level of commitment to a project's maintenance and evolution. At the core of the "onion" lie contributors with write access to a project's source code repositories (referred to as *core developers*, or committers); they have the highest level of access to the project thus can introduce changes to the code directly, but also share the greatest responsibility of delivering and evolving a viable product. The next smallest layer comprises *peripheral developers*, who typically propose smaller changes, known as *patches*, in the form of bug fixes, feature improvements, or contributions to documentation; patches are reviewed by core developers and are added to the project's source repositories at their discretion. Finally, *users* are the downstream consumers of OSS; their participation in OSS communities is mostly restricted to discussions on mailing lists or issue reports.

OSS communities are also faced with high turnover [7]. Therefore, to ensure a community's sustainability over time, the progressive integration of new members in all layers of the "onion" is paramount [8, 9]. This process, typical of meritocratic communities [10–12], has received a lot of attention in the empirical software engineering research literature, where it is variously referred to as developer initiation [13], entering the circle of trust [13], migration [14], or immigration [15]. A typical trajectory to becoming a core developer is to start communicating with other project contributors 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, for example, submitting patches, or working on bug fixing activities [8, 15]. Eventually, developers may reach the core of the "onion" through the recognition of their contributions.

The congruence of a newcomer's social and technical activities is crucial for successful participation in an open source project [16]. In a sense, a newcomer is as integral to the project as their contributions, be they communications or bug fixes. Strong patches containing working and well-tested code lead to increased trust in the developer's ability to add technical value to the project. Similarly, strong social skills signify that the developer can integrate well with the project team. The more trustworthy a developer, the more likely she is to eventually reach the core of the "onion" and achieve committer status. Generally, only those contributors who have sufficiently proven themselves through their activities become committers [10–12].

Developer initiation in OSS, thus, depends on the social and technical actions of project contributors, *e.g.,* who they talk to, the number of social links they develop with other project members, their communication patterns, patch submission activity, or bug identification and fixing activity. But to what extent do social activities and technical activities work together to increase one's chance of advancing through the ranks? And how does one's socialization behavior evolve from before to after having been recognized as core developer or committer?

In this article we revisit the issue of migration between roles in OSS projects, studying it from a social network analysis perspective. We collect data about *social* (*i.e.,* communication on mailing lists) and *technical* (*i.e.,* patch submissions and code changes) activities of developers in 6 projects from the Apache Software Foundation (ASF), arguably the most famous example of a large-scale meritocratic community [10–12]. From the data we build and compare statistical predictors for the likelihood of a newcomer to become a core developer (committer).

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 messages they *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. We find that social-links-based models exhibit better predictive ability for developer initiation than models incorporating patch submissions.
- Whether a contributor will eventually become a committer 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, yield good prediction results that are within 10% of the accuracy of models learned on three months of data. In other words, as little as one month of trace data is sufficient to predict whether a contributor will reach the team core (committership).
- Contributors steadily ramp-up their social activities in each project, on average, before becoming core developers. After obtaining committer status, their social behavior is more individualized, falling into few distinct modes of behavior. In a significant number of projects, immediately after the initiation there is a notable social cooling-off period. Interestingly, and perhaps predictably, there is an apparent robustness in the communication patterns of developers with prior engagement in the ASF community, which we expound on in a case study.

– The impact of both social and technical participation declines with project age. In other words, it becomes more difficult to attain committer status as the project matures.

These findings have implications for researchers and practitioners alike. On the one hand, our work contributes to the growing body of research in empirical software engineering (*e.g.,* [17–21]) advocating the importance of *social* factors in the evolution of (OSS) software projects. On the other hand, OSS practitioners, be they newcomers wishing to contribute to a project, or managers interested in the sustainability of a project's community of contributors (*e.g.,* for Apache, Project Management Committees, whose role, among others, is to "further the long term development and health of the community as a whole"[1]) can use the evidence we provide to optimize their initiation or oversight processes, respectively.

The rest of the article 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.

1.1 Background: The Apache Software Foundation Process

The Apache Software Foundation (ASF) community has a flat, bazaar-like [22] structure, in which "anyone can be a contributor"[2]. Still, different contributor roles exist, and are clearly defined[3]. Similarly to the "onion" model, in ASF one can distinguish between *users* (as in the "onion" analogy), who contribute to discussions on mailing lists or report bugs, *developers* (peripheral developers), who in addition provide patches or contribute to documentation, and *committers* (core developers), who were granted write access to the code repository and can push their changes directly. In addition, ASF committers can be elected based on merit to participate in each project's *Project Management Committee* (PMC), the entity which, as a whole, controls the project and approves active developers for committership. Finally, active committers or PMC members can be elected as *ASF members*. They are considered the "shareholders" of the foundation, with project-related as well as cross-project responsibilities and activities, such as electing the board or proposing new projects for incubation.

Apache projects as well as many other OSS projects adhere to the meritocratic governance model, in which participants gradually gain influence over a project, and consequently advance through the ranks, through the recognition of their contributions. In this article we focus on the *transition from developer to committer*, often studied in the empirical software engineering research literature [13–15]. To become a new committer, a contributor must first gain acceptance within the community and show *commitment* to the project. This can be accomplished through any number of ways—assisting users on the user list, testing code, writing documentation, bug triaging, or writing code and submitting patches for review and integration into the code base. Only when a developer has contributed sufficiently

[1] `https://www.apache.org/foundation/how-it-works.html#pmc`

[2] `http://community.apache.org/contributors/`

[3] `http://www.apache.org/foundation/how-it-works.html#roles`

to a project, they may be nominated for committer status by an existing committer, after which voting takes place. Existing committers may then choose to grant this developer committer status, allowing her to make direct changes to the project's source code repository. Sought-after characteristics of ASF committers are "the ability to be a mentor and to work cooperatively with [one's] peers" [4] Such traits can be observed early on, during a developer's *technical* and *social* collaborative activities before obtaining committer status. Technically, 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 developer understands the software at a technical level. To have their patches accepted, developers must submit work of high quality (*e.g.,* well tested, well integrated with that of others) as well as advertise and argue its relevance to the project. Socially, actively contributing to mailing list discussions and offering useful suggestions and criticisms are key ways in which a new contributor can attract the attention of existing committers and gain social reputation within the community.

Hence, whether a developer will eventually become a committer is a function of all their social and technical activities within the project's ecosystem. In general, trust in a developer's technical and social skills is believed to increase with time, as a result of increased contribution and interaction with other contributors [23]. To illustrate this process, consider the example of John (name redacted for privacy reasons), whose first public interaction with the Apache Pluto community is on the mailing list in August 2006:

> *Hello all, I'am John from the University [...], we are developing the Prototype for the JSR 286. I hope that we can discuss the code [...] we have made and then develop new code for Pluto together [...],*

referring to his and some of his fellow student's intentions to contribute to Pluto. John gets the attention of Pluto committers and is immediately welcomed as a developer into the community, but without commit rights:

> *John, who already subscribes to this list, appears to be cooperative. He wants to work with us and contribute code to our SVN repository with the help of current Pluto committers. He already has agreed to work on two of the most important container-related Pluto 1.1 issues. I'm suggesting we create a branch based on our current Pluto 1.1 trunk. He and his group can submit their code by creating issues in this Jira branch and attaching patches to these issues. It will be up to the Pluto committers to add these contributions to SVN. In time, I hope that John can earn the right to be a committer himself [...].*

After submitting a total of 33 patches, John signals his readiness for committership in March 2007:

> *[...] Maybe we could all benefit if I can commit our patches myself. Please write me comments if we made anything wrong and how we can make it better.*

However, it seems too early for Pluto committers to trust him yet:

> *I understand your frustration. At the same time, I encourage you to continue to be active on this list. The development list is a HUGE part of the open source community, and you will only help your cause by having these types of discussions as well as airing design decisions on this list. That type of consistent communication will get you a long way - not only in getting attention from committers, but also in obtaining commit credentials [...] I would also recommend that your group begin to*

[4] `http://community.apache.org/contributors/`

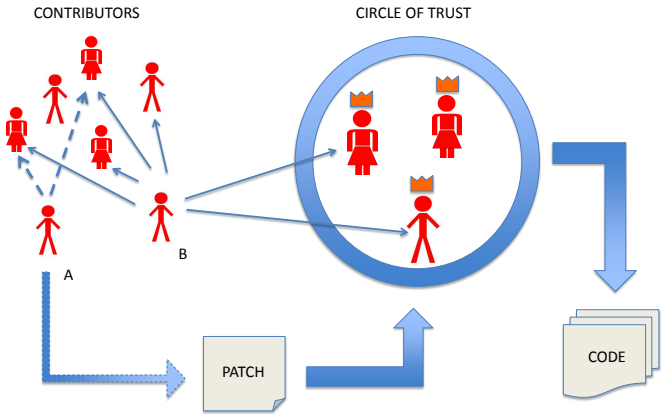

Figure 1. We consider an ASF project community's circle of trust to be comprised of committers. Developers A and B are candidates for entering the circle. The activities of B are more significant positive predictors of her gaining committer status because she communicates more (thin solid arrows) than A does (thin dashed arrows), even though A contributes patches and B does not.

> *communicate exclusively on the mailing lists. This will show that you are a critical part of our community. Becoming a committer is as much (if not more) about community involvement as it is about code.*

John presses forward and eventually becomes a Pluto committer in August 2007, one year after joining the mailing list.

Therefore, temporal measures of participation (both technical and social), including the number of emails sent, one's degree in the email social network, the number of committers among a developer's neighbors, the numbers of bugs reported, or the number of patches submitted, may all be indicators of the community's trust in a developer's abilities [24, 25]. Different paths to committership may exist, as illustrated in Figure 1. There, *A* contributes patches, but communicates rarely with other project members, while *B* does not submit patches but communicates extensively with other project members. Our results, presented in this article, show that *B*'s activities are more significant for predicting her future committership than are those of *A*.

It is important to note that this is just the basic framework of how a contributor becomes a committer in ASF projects, and the situations may vary for different projects, or different stages of evolution of different projects. For example, some projects may require developers to enter bugs into an issue 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, developers and committers in those projects, thus affecting the dynamics of earning trust. Additionally, different contributors may have different motivations to join a project [5,26,27], *e.g.,* enjoyment, reputation building, and skill improvement. The commercial backers of some OSS projects may also provide incentives for skilled programmers to contribute in order to grow their project in its 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 [26].

1.2 Research Questions

In this article we seek to identify effective social activity predictors of developer initiation in OSS projects, with a focus on ASF projects, and to improve upon existing models for predicting future committers. Previous work has focused mainly on technical activity predictors of developer initiation (*e.g.,* based on code commits, patches submitted, or issues resolved). We hypothesise that social activity yields at least as strong predictors as technical activity, especially in OSS communities such as Apache, which "values the community more than the code"[5].

First, we ask which social metrics are effective predictors that a contributor will become a committer, and how do they interact with the technical measures of patch activities?

Research Question 1: To what extent can developer initiation in OSS projects be modeled as a function of patch activities and social communication? And to what extent solely as a function of social communications?

Early in their tenure as OSS project contributors, people's patterns of technical and social activities are rapidly changing. Contributors 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 (or may not) be unreliable early in a person's tenure. Here, we seek to predict project participants' likelihood of becoming a committer based on the patterns of their activities early in their tenure, and in doing so understand how early can predictions be done, with reasonable reliability.

Research Question 2: How accurately can developer initiation be predicted from their earliest activities in the project? That is, can we tell if someone will become a committer based on their activities in the first three months? Six months? Or as little as one month?

As with many deadline oriented tasks, approaching the goal is usually associated with the anticipation of it. In that regard, we would expect not much variance in people's behavior just before being initiated a developer, as one tends to ramp up one's activities to reach a goal. The communication activities that follow the initiation time would presumably be less cohesive, and more connected to individual preferences and styles of work. This may be more apparent for people who already participate in other OSS projects. Whereas in the RQs above we contemplate the importance of social communication to achieving committer status, next we ponder at a finer resolution how the social activity of future committers changes as they approach their developer initiation time, and how their communication patterns evolve hence. Specifically, we ask,

[5] `http://community.apache.org/contributors/`

Research Question 3: What is the relationship between the amount of one's communication activity and the periods of time just before and just following their initiation as committers? Do they become more social as they approach their initiation time? Do they fall back into more individualized patterns after? Is their behavior related to their experience in other projects?

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 committers in a project typically grows proportional to its size, the number of participants in a project's mailing list (*e.g.,* developers and users) grows exponentially. This increasing gap between potential committers and new position openings in turn change how trust is earned as a project matures.

Research Question 4: Is it easier or more difficult to become a committer later in the project?

2 Related Work

In this article we model developer initiation and study social interactions between contributors to six Apache Software Foundation projects. References to related work pertaining to individual steps in our analysis process can be found throughout the text. In this section we discuss three other areas of related work, studies of ASF projects in Section 2.1, studies of developer initiation in OSS in Section 2.2, and general studies of newcomer incentives and socialization within an organization Section 2.3.

2.1 Studies of ASF Projects

The Apache Software Foundation has a rich and public history that has driven many research studies. The number and scope of these projects is too large to give full justice here, we mention just a few to illustrate the variety of contexts in which these projects have been studied. Bird *et al.* used Apache software in his work on social network mining [28]. Rahman *et al.* used Apache projects to study cross project defect prediction, and bias in software engineering datasets [29, 30]. Jureczko and Madeyski also studied cross project defect prediction using several Apache projects [31]. Jureczko and Spinellis used a collection of Apache projects to model defection prediction using Object Oriented metrics [32]. Posnett *et al.* used 18 different ASF projects to illustrate the risk of ecological fallacy in empirical software engineering studies [33].

2.2 Studies of Developer Initiation in OSS

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" [5, 6].

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 [34]. Ducheneaut examined a single individual and his process of promotion to a core developer in the Python project [8]. Jensen and Scacchi studied role sets and the process of role migration in Mozilla, ASF and Netbeans [14].

Sinha *et al.* studied how developers enter the "circle of trust" by identifying key factors that lead to committer status [13]. 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 committer 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 [35]. Shibuya and Tamai performed case studies and confirmed these findings on other OSS projects (GNOME, OpenOffice.org, MySQL) [36].

Qureshi and Fang have identified different classes of committers 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 [37].

Bird *et al.* quantitatively modeled the relationship between time spent with the project and the probability of becoming a committer; their model used patch activities, social network attributes, and the time to first commit from the time of first communication on an email network [15]. Using proportional hazard rate modeling they observed that a committer'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 committer 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 committer is in contrast to our work in that we focus on identifying "who" is more likely to become a committer 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 [38]. 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.

2.3 Genereal Studies of Newcomer Incentives and Socialization

The dynamics of social activity patterns around times of precisely defined goal achievements have been studied formally before. In a broader context, the issue is related to reward sensitivity, "an incentive motivational state that facilitates and guides approach behavior to a goal" [39], also referred to as goal-directed approach behavior [40]. Particularly interesting are the works of Lucas *et al.* [41] and Ashton *et al.* [42] expounding the importance of reward sensitivity for explaining one's social behavior. In addition, goal-directed approach behavior has been studied in the context of gamification [43]. Cheng and Vassileva [44] observed "gaming" behavior in an educational online community, *i.e.,* incentivizing user contributions with status enhancements (analogous to incentivizing developers with committership prospects in our context) motivates users to adjust their behavior to maximize their chances of receiving the reward, even inappropriately by adding low-quality resources. Farzan *et al.* [45] found that some incented users stop contributing after reaching a specific status level in a social networking website for employees at IBM. Anderson *et al.* [46] studied badges, or online distinctions achieved by users as reward for activity, and their role in steering online users behavior in Stack Overflow. They found that "activity on the targeted actions increases substantially before users achieve the badge, and then almost immediately returns to near-baseline levels. Most of the other site actions are not adversely affected—the rates of these actions remain relatively stable over time." Similar findings have been reported by Grant and Betts [47].

While our results are specifically about achieving committer status within open source projects, it fits within a more general framework of understanding how newcomers are integrated into their work environment. Begel and Simon studied newcomers to software development within Microsoft [48]. Their focus was on understanding how science pedagogy prepares students for the workforce. One finding of interest to this study is that they found that newcomers spend significant portions of their time communicating with peers. The process of transitioning from outsider to insider is called *Organizational Socialization*, or *Onboarding*, in the psychology literature [49, 50]. Baur *et al.* study the orgnaizational socialization of 70 individuals using path modeling to ascertain how the effects of role clarity, self-efficacy, and social acceptance, mediate the effects of organizational socialization tactics and newcomer information seeking [51]. A finding of interest here is that social acceptance has a positive mediating affect on newcomer information seeking with respect to performance, organizational commitment and intentions to remain.

3 Data Gathering

The Apache Software Foundation is an umbrella for hundreds of different OSS projects. We sought to analyze a diverse sample of them, with respect to both project size and activity. We selected six ASF projects, each with an acceptable minimum number of contributors, to make our modeling results statistically acceptable. The six projects together with summary statistics are presented in Table 1.

For each project we mined data from three sources. The first is the *developer mailing lists*, which contain traces of developers *social activities*, from which we

Table 1. The ASF projects selected in this study show diversity both in size and in relative activity. #Users refers to the number of individuals in the email social network. #Committers refers to the number of distinct committers to each project's source code repository.

Project	#Users	#Committers	#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

reconstruct email social networks. We deemed the mailing lists provide an unbiased set of communications, sufficient for our purposes of capturing neccessary data for our statistical predictive models[6]. The second is the *issue tracking systems*, which contain information about patch submissions, indicative of one's *technical activities* pre-committership. And third, we mined the *source code repositories* to extract information about committer status. All sources were mined from the earliest date the data was available, until the date of mining (March 2012). The start and end dates reported in Table 1 represent the intersection of available data from all three sources, for each project.

In this section we describe the specific process we followed to extract data from each of the three sources. An overview is depicted in Figure 2. All data (processed and raw) described here along with all the scripts used to process them are available in an online appendix, at `http://csiflabs.cs.ucdavis.edu/~ghareh/supplementary/oss/`.

3.1 Unmasking Aliases

OSS contributors often use different aliases (combinations of names and email addresses such as `<John Smith, smith@gmail.com>`, `<Smith, John@smith.com>`, `<John S., J.smith@ucdavis.edu>`) in different repositories they participate in (*e.g.,* source code repositories, issue trackers, mail archives), or even in the same repository but at different times (*e.g.,* an ASF committer with a personal, say `gmail.com`, email address might also push changes from an account configured to use her `apache.org` email address). Since such aliases represent a single physical entity (*i.e.,* a person), they must be merged if one is to accurately capture a contributor's total activity within the project.

Unmasking aliases (or identity merging) is a well-recognized problem in the literature (*e.g.,* [28, 52, 53]), thus far without any perfect solutions [54]. In this article we employ an extended version of the technique developed by Bird *et al.* [28], based on heuristics and string similarity measures. The heuristics include *guessing* a person's likely email address prefixes based on their first and last names (*e.g.,* John Smith might use prefixes such as `john`, `jsmith`, `johns`, or `john.smith`), then using this information to aid matching the different aliases. Our extension consisted in refining the heuristics (*e.g.,* we have observed in our dataset that certain

[6] Issue trackers also capture communication between committers and developers. We did not use those because the mailing lists contained a large enough communication sample which was not obviously biased in any way

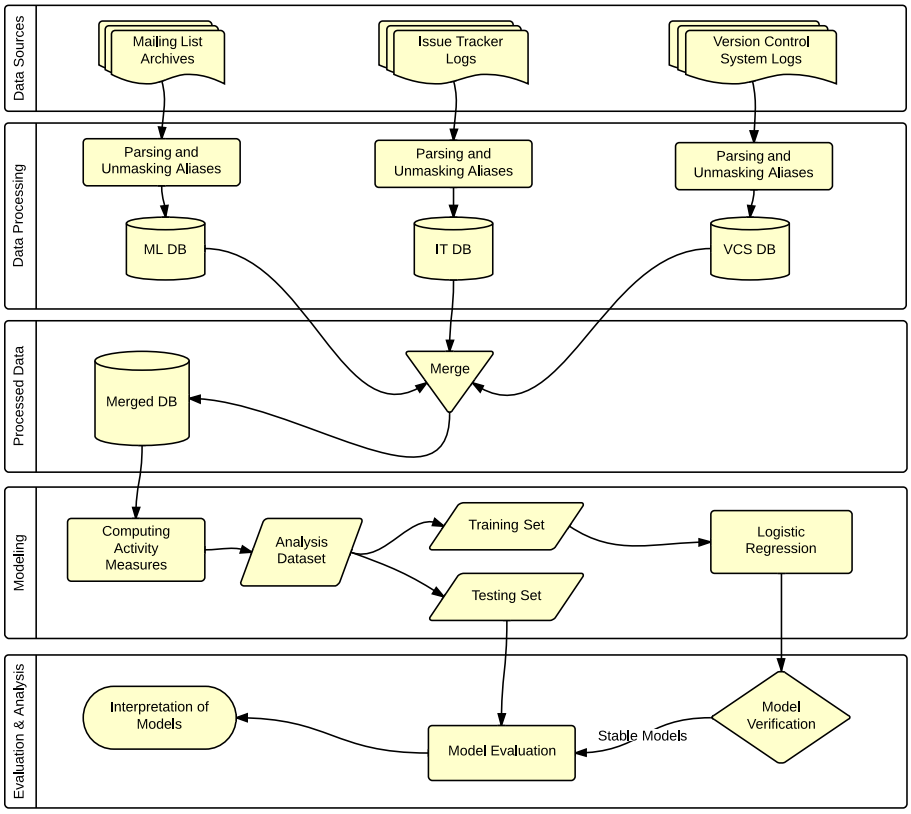

Figure 2. The process diagram of data gathering, processing, modeling, and finally evaluation as described in the paper.

email addresses such as `jira@apache.org` may be used by different contributors; therefore, such addresses should not be used for matching) and automating the procedure further, to reduce the need for human interaction.

Succinctly, the process consisted of the following steps. First, we filtered names and removed all suffixes, prefixes, and generic names, such as Dr., Mr., Jr., or Admin. Then, for each pair of aliases we calculated a similarity score (on a unit scale) based on the Levenshtein (edit) distance between the different alias parts (name-to-name, or name-to-email-prefix), as described in the original work by Bird *et al.* [28]. Next, we merged perfect matches (having score 1) automatically, and presented less than perfect matches that achieved a score of at least 0.93 (threshold determined empirically on other Apache data to offer a good tradeoff between false positives and false negatives) to one of the authors to disambiguate. Finally, the results were reviewed by a different author and few incorrect matches were corrected.

The process was repeated for each of the three mined data sources (source code repositories, issue trackers, mail archives), with more manual intervention of the authors to merge aliases *between* the three.

3.2 Constructing Email Social Networks

It is a common 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 [28]. While other venues of communication in OSS exist (ranging from discussions around issues recorded in bug tracking systems, to IRC channels, to private email or even offline communication between developers), developer lists are known to be the hub of developer communication in OSS [55], *i.e.,* the place where most communication happens and where the team meets to discuss issues, code changes, additions, etc. Therefore, we view developer lists (such as `dev@ant.apache.org` mined in this article) as representative of the different developer communication media available in OSS. In addition, developer lists are very suitable for studying developer initiation in ASF since all proposals and voting for granting developers committership are recorded therein.

Any message sent to a mailing list will be broadcast to all subscribed participants. Still, point-to-point exchanges (links) between different list subscribers can be inferred, using the reply information [28, 56, 57]. When person B replies to a message originally broadcast by person A to the list, there is a high chance that B primarily intended to communicate directly to A as opposed to again broadcast to the entire list [28]. Such links are in fact two-way social links, in that the communication occurs both ways (A communicated with all list subscribers hence implicitly also to B; B communicated explicitly to A since she replied directly to A).

The networks resulting from parsing the emails in each project's mail archives, reconstructing the discussion threads and extracting the in-reply-to links are known as Email Social Networks (ESNs) [28]. ESNs are indicative of a project's social structure and can be used to assess a contributor's *social* activity. We created ESNs for each of the six projects by removing self-loops, *i.e.,* replies to one's own message, and allowing multiple edges between people (*e.g.,* as inferred from communication at different times). We use the ESNs to compute social activity measures for each developer, such as the total number of messages sent within a time unit, the number of neighbors, or the number of neighbors having committer status, as detailed in the Modeling section below.

3.3 Mining Patch Submissions

Patches are small pieces of code intended to fix bugs or otherwise incorporate small changes into the code base. While patch submission is not limited to non-committers, the term "patches" is typically used to denote code contributions by developers who have not yet been granted commit rights to a project's source code repository. Patch submission does not imply patch acceptance, with committers reviewing and applying patches as they see fit. Patch submission does however signal one's interest to contribute to a project, and is a typical *technical* means for a developer to build reputation within a community before reaching committer status.

There are multiple methods and formats in which one can submit a patch to an OSS project. The commonly employed format for a patch submission is a "diff" file containing the proposed changes. Patches are submitted for review

```
bool isMessageAPatch(message m) {
    messageText = getMessageText(m);
    diffExpressions = ["+++", "---"]
    if (contains(messageText, diffExpressions)==T)
      return(TRUE);
    attachments = getMessageAttachments(m);
    for (attachment in attachments)
      if (isAttachmentAPatch(attachment) == TRUE)
        return(TRUE);
    return(FALSE);
}

bool isAttachmentAPatch(attachment A) {
  patchExpressions = [".patch", ".diff", "patch."]
  if (contains(fileName(A), patchExpressions) == T)
    return(TRUE);
  else
    if (isArchive(A) == TRUE) {
      files = extract(A);
      for (file in files)
        if (isAttachmentAPatch(file) == TRUE)
          return(TRUE);
    }
  return(FALSE);
}
```

Algorithm 1: The pseudocode for mining patches from message texts and attachments

either as attachments to issues opened in the project's issue tracking system (*e.g.,* Jira, Bugzilla), or as attachments to messages posted on the developer mailing list. Furthermore, since patches are in fact source code fragments, they can be embedded directly in the text of a comment posted to an open issue or, similarly, in the text of a message sent to the developer list. For ASF projects, all these methods for patch submissions are common.

File names of patches submitted as attachments typically end with a `.diff` or `.patch` extension. This convention is not always respected, as we also found patches submitted under different names (*e.g.,* `patch.txt`), or multiple patch files combined in an archive. To capture this variance, we developed a set of heuristics based on regular expression pattern matching, that included inspecting the contents of archived files. Similarly, we used regular expression pattern matching queries to extract inline patches from emails, or comments or descriptions to issues reported in Jira and Bugzilla. A pseudocode representation of the processes used to mine patches is given in Algorithm 1. The distribution of patches mined in each project from the different sources is presented in Table 2.

3.4 Mining Source Code Repositories

A source code repository and a version control system, *e.g.,* Git, SVN, and CVS, facilitate collaboration among committers by maintaining a history of changes and an associated log entry for each change. These systems can provide various information about a project's size (*e.g.,* a list of all the files), team size (*e.g.,* a

Table 2. Different projects have different practices for patch submission, making extraction challenging.

	Maling Lists		Bugzilla	Jira	
	Inline	Attachments	Attachments	Inline	Attachments
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

Table 3. The total number of committers to each project, out of which those that do not appear in the ESN or have less than three months of social activity data available prior to them becoming committer (the *filtered* column). Remaining committers are what we consider *initiated developers*.

Project	#Committers	#Filtered Committers	#Remaining Committers
Ant	44	13	31
Axis2_c	24	3	21
Log4j	18	8	10
Lucene	41	9	32
Pluto	24	10	14
Solr	19	5	14

list of committers), or a detailed record of all changes induced by any committer. Currently, all six ASF projects considered here use Git as their version control system, but some initially used either CVS or SVN and later migrated to Git. The current Git logs incorporate the history of changes to these projects prior to their migration to Git.

Among others, this information allows us to distinguish between *developers* (contributors that are part of the ESN and submit patches through any of the methods described above) and *committers* (contributors who push changes directly to the project's source code repository as evident from the version control logs), at any time in the history of a project. Consequently, we consider the date of one's first recorded commit in the project's version control logs as their date of becoming a committer. While this is an approximation, since a contributor may have been granted committership before her first actual commit, both previous work [15] as well as manual inspection of a random sample of developers in our dataset suggest this to be an accurate representation of one's initiation date.

Table 3 lists the number of distinct committers (after unmasking aliases) for each project, as well as the number of distinct committers used in the statistical modeling (described next), *i.e.,* those present in the project's ESN and for which at least three months of social activity data is available prior to becoming committers.

3.5 Computing Social and Technical Activity Metrics

Gathering the data from the three sources described above resulted in a rich longitudinal dataset of social and technical activities in each project. Figure 3 provides an illustration of this dataset, with the unfolding of events in one project involving three contributors A, B and C. Assuming the project started at time t_1, we

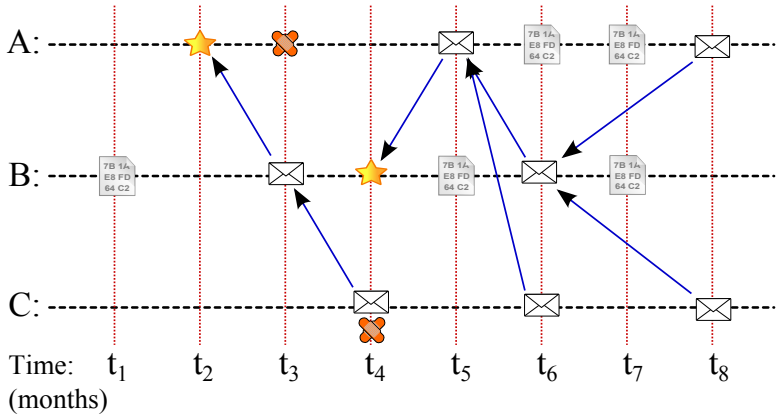

Figure 3. Illustration of the interplay between social and technical activities in our dataset. The metrics for this sample are given in Table 4. A star denotes starting a new thread, the envelope symbolizes responding to a thread/message, the patch icon denotes submitting a patch and the code icon symbolizes committing code to the repository. An arrow from A to B denotes that B's message was sent in response to A's. The horizontal axis denotes time.

Table 4. Social and technical activity measures for sample dataset in Figure 3 assuming $k = 3$.

ID	Initiated	Num. Patches	Num. Messages	Num. Threads	Neighbors	Neighbor Committers	Project Age
A	TRUE	1	5	1	2	1	90
B	FALSE	0	6	1	2	1	90
C	FALSE	1	3	0	2	2	120

observe that B commits to the repository during the same month (hence B starts off as a committer). At time t_2, A starts a new discussion thread on the mailing list. At time t_3, B replies to the thread previously started by A. Shortly after, A attaches a patch to an issue reported by someone else in the issue tracker (not shown). At time t_4, C joins the discussion thread started by A at t_2, by replying to the email sent by B at t_3 and attaching a patch to this reply email. We skip ahead to time t_6, when A's first commit is recorded in the version control logs, hence t_6 becomes A's initiation date as committer. In contrast, C continues being active on the mailing list until t_8, the end of time displayed, but without ever being initiated as committer.

Note that we will build different prediction models for one's likelihood of becoming a committer (time-independent binary outcome variable) using data about their communication and patch submission activities collected during their first k months of activity in the project (we experiment with $k = 1$, $k = 3$, or $k = 6$, recall RQ2). This implies that a developer's communication or patch submission activities must have preceded their initiation date as committer by at least k months, otherwise they would be excluded from the sample. For example, in Figure 3 B would be excluded at any of $k = 1$, $k = 3$, or $k = 6$, while A would be excluded at $k = 6$ because her activity started at t_2 and her initiation date is t_6.

On the dataset resulting from the above pre-processing, we compute the following measures of social and technical activity for each developer (for the example in Figure 3, the measures are given in Table 4):

- *Initiated*: A binary variable indicating whether this developer ever committed directly to the repository *after* her first message in the mailing list.
- *Number of Patches*: The total number of patches submitted during the first k months of activity in the project (months are approximated as 30 days).
- *Number of Messages*: The number of edges connected to a node in the ESN, *i.e.,* a node's degree. This is not just the number of messages one sends, but rather the number of *replies* one **sends** plus the number of replies one **receives**.
- *Number of Threads*: The number of threads started. A thread is a message that is *not* sent in response to any other messages.
- *Neighbors*: The number of unique nodes that a node is connected to in the ESN. This is different from *Number of Messages* in that a node can connect to other nodes through multiple edges.
- *Neighbor Committers*: The number of unique nodes with commiter status that a given node is connected to in the ESN.
- *Project age*: The number of days from the start of the mailing list to the first appearance of this node, *i.e.,* the first message received by or replied by that person in the ESN. In our example both A and B's *Project age* are 90 days because as explained previously, only response messages are counted towards the ESN and for A it is the first response it gets that counts as its first link in the ESN.

4 Modeling

In this section we describe our methodology, including logistic regression and the analysis of socialization dynamics around the time of becomming a contributor.

4.1 Modeling Committer Initiation

We use logistic regression, a generalized linear modeling technique designed to model probabilities for dichotomous outcomes, to model whether or not a developer will reach committership based on several social explanatory variables.

The logit $log(\frac{1}{1-p})$ models unbounded response as a probability using *maximum likelihood* estimation which, given a distribution, finds the values of the parameters that give the observed data the greatest probability. That is, maximum likelihood is used to estimate the following general model which yields an estimated probability $\hat{p_i}$ that the true value of the response is 1 [58].

$$log(\frac{1}{1-p}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon$$

For each predictor the *z-test statistic* is computed. This statistics divides the estimated value of the parameter by its standard error and is used to assess the significance of the variable. This statistic is a measure of the likelihood that the actual value of the parameter is not zero [58].

Previous work in this area has used survival modeling to model the trajectory over time of developer initiation [15]. 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 the *"Initiated"* outcome. We generated multiple models for each of the research questions. The variables used for each of the models are described in the results.

For our models we are primarily interested in the direction of the effect of each predictor. We want to control as much as possible for other sources of variation that might be incorrectly attributed to the variables of interest.

The age of the project or "Project age", *i.e.,* when a person joins the 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 [58]. Since all untransformed values in our data are skewed, the increase in the value of a variable by half a 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 1.5 is much more meaningful than changing from 100 to 100.5.

For transformed variables, comparison with the non-transformed variable shows that this transformation yields a better fit using Vuong's non-nested test [59].

Considering we are trying to predict who *is going to* become a committer, sample data for developers who were initiated in fewer than k months were removed from the dataset (Table 3).

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. One way to address this issue is to merge all projects' datasets into one dataset. This raises several concerns such as scaling of variables *i.e.,* 30 messages may considered much in one project while it is a small value in another. At the same time multiple project dependent parameters that we cannot measure such as project "culture", or simply we are not aware of exist. The other solution is to look at the stability of each model coefficient's statistical significance, as determined by the coefficient's *p-value*. We choose the latter because it does not pose the aforementioned threats.

Commonly, a p-value of less than 0.05 is an indicator of significant results. If a predictor's coefficient is statistically significant across all projects, then it is more likely to be a project independent factor. On the other hand, if a coefficient is only statistically significant on few of the projects, it is most likely dependent on some project parameter, such as "culture", and "maturity".

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 article, 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 article. We then move towards evaluating each model's performance *i.e.,* predictive power.

Some of the basic definitions of a binary classifier's performance are True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). True Positive/Negative is the number of positive/negative samples that the clas-

sifier guessed correctly. False Positive/Negative is the number of negative samples that the classifier guessed wrong. TP rate is defined as TP over number of all actual positive samples in the testing set and FP rate is defined as FP over the number of all actual positive samples. The Receiver Operating Characteristic or the ROC curve illustrates the performance of a binary classifier in a *FP rate-TP rate*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 because it will always have a TP rate of 1, and an FP rate of 0.

To evaluate a model's predictive power, we use the *Area Under the Receiver Operating Characteristic (AUROC)* measure [58]. For each model, we measure its AUROC and the closer this value is to 1, the better the predictor will be. In cases where we aggregate performance of many models, we simply measure the mean of the models' AUROC.

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 committers vs. 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 committers to 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.

4.2 Dynamics of Social Activities Before and After Becoming a Developer

Here we look at the dynamics of the amount of socialization, or social activities, *i.e.,* emails, of developers around the time of them becoming a committer. We start by counting the number of messages sent to or received by a person in the 6 months prior to their developer initiation and the 6 months immediately after the event. We filter out those developers who are not active in at least 6 months before and after, to avoid biased results (the number of remaining developers is given in the results). Then, we can observe the overall behavior of the developer population by comparing the number of messages before and after becoming a developer. By aggregating the data in buckets we can statistically assess how socialization changes from the period before to the period after initiation. In addition to the statistics we also conduct a case study on a handful of developers to offer email content-based evidentiary support for our findings.

5 Results and Discussion

5.1 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

Table 5. 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

question with Figure 4. This figure shows that, although most of the time there is a meaningful difference between committers and developers who submit patches, still there are many developers who behave like committers 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 5). Based on what was described in the methodology section the prediction model's formula in this case is:

$$Initiated \sim \log(project\ age + 0.5) + \log(number\ of\ patches + 0.5)$$

In Table 5 (as well as all other tables with same format hereafter) we have shown the results of logistic regression on each projects data separately. Each row below each project represents that specific predictor. The "Estimate" column represents the estimated coefficient for that specific predictor. "Std. Error" is the standard error of the coefficient and "z value" is the t-statistic of the coefficient. "$Pr(> |z|)$" shows the p-value of the coefficient for the predictor.

However, adding "number of messages" (as an indicator of social collaboration) to this model results in patches slightly losing their significance (Table 6). 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

Table 6. The second logistic regression model, adding "number of messages" to the previous model: *Initiated* $\sim$ log(*project age*+0.5) + log(*number of patches* +0.5) + log(*number of messages*+0.5). 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

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 Figure 5. 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 AUROC, 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 AUROC, 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.

Furthermore, we attempt to 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,

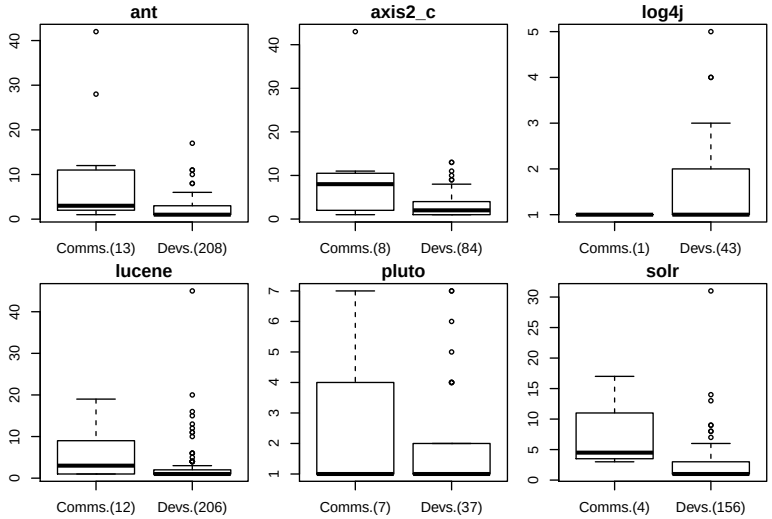

Figure 4. Distribution of number of patch submissions by committers and developers in each project. Only individuals with at least one patch submission are plotted, since adding those with no submission would highly skew the plots towards zero. The numbers in the parentheses show each group's population (after filtering non-patch-submitters). 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 committers and those who don't when measured in the first three months of participation.

Table 7. 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. comm. neighbors	**0.62**	**0.52**	0.49	**0.55**	**0.53**	**0.65**
neighbors vs. comm. 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. comm. 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**

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 7). 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 8, most are hampered by the high variance inflation factor owing to the high correlation between "Number of Messages", "Number of neighbors", and "Number of developers" (Table 7). "Number of threads", however, was significant in two projects, Ant, and Solr, and had

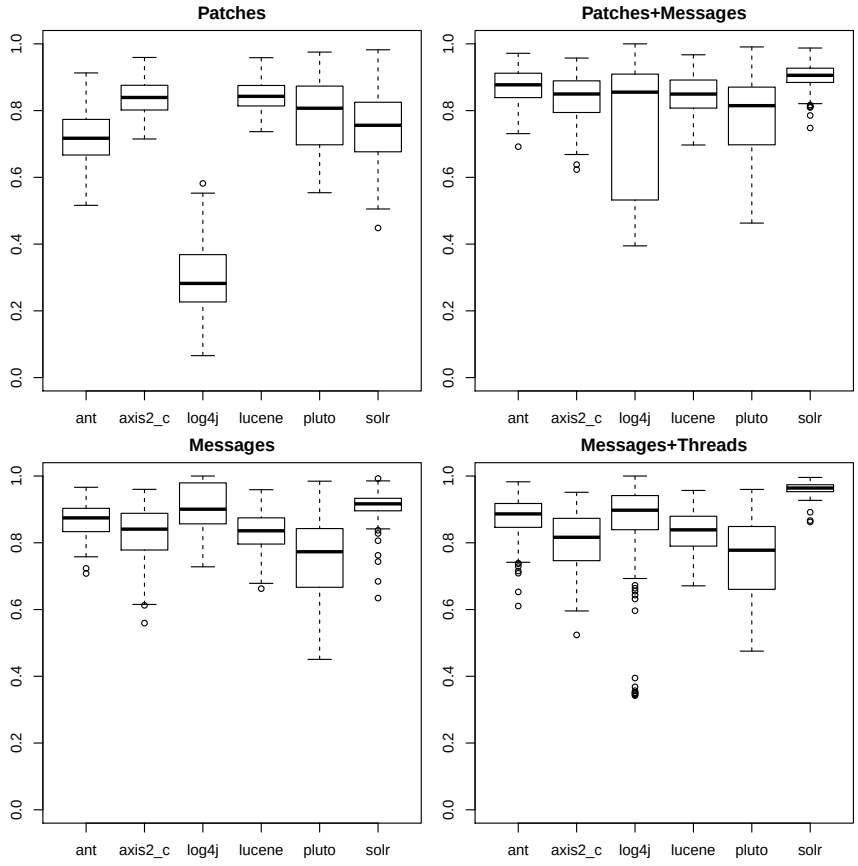

Figure 5. Social measures outperform patch submission in a predictive setting. AUROC of 4 models, on 250 iterations of modeling using stratified data.

a 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".*

5.2 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

Table 8. 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**

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?

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

A limitation in evaluating models for long 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). Choosing time windows of 1 month to 6 months for observing participants' activity will still include more than half of the developers in the dataset.

Table 9. Social metrics yield better performing predictive models for developer status across most projects. Mean AUROC 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 AUROC 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 + Messages	Messages	Messages + 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**

Table 10. 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 a 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**

The modeling results can be seen in Table 10. 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 Figure 6 that the predictive powers of the models differ slightly from one time window to another. Time windows less than 3 months slightly suffer from lower 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. Figure 5 (bottom) and Figure 7 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.*

5.3 Research Question 3

Previous studies on gamification and goal-directed approach behavior, reviewed in Section 2.2, suggest viewing developer initiation as an incentives-based process, in which uninitiated developers are motivated by obtaining committership. In this

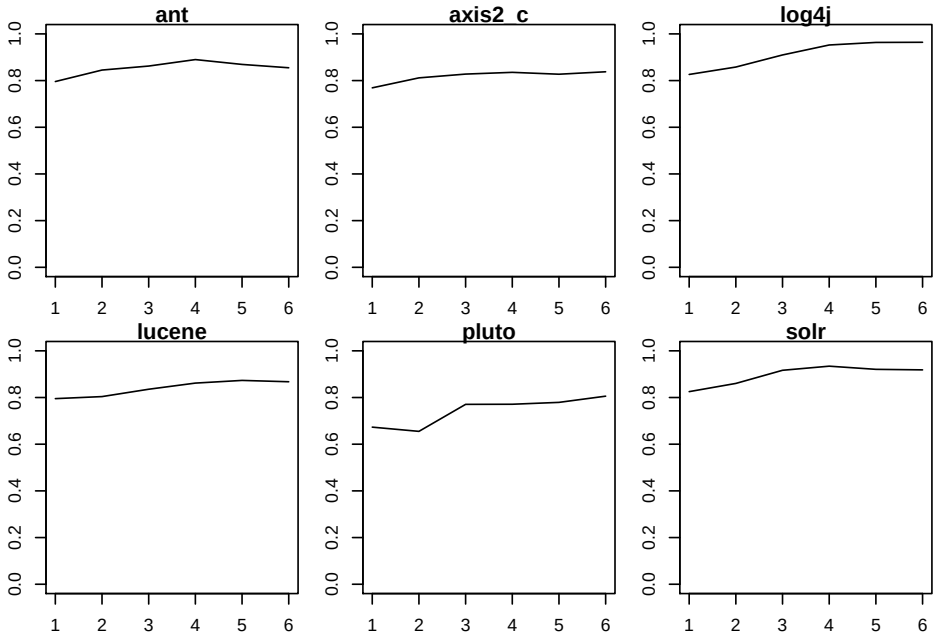

Figure 6. The predictive power of the model using "number of messages" from 1 to 6 months, each on 250 iterations of modeling using stratified data. The AUROC for each project slightly improves until 3rd or 4th month, and then stabilizes.

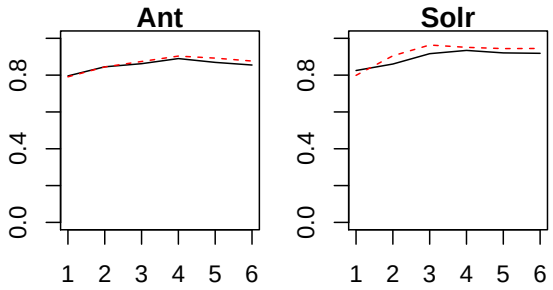

Figure 7. The AUROC of two projects with two different models. Black lines represent models using only "Number of messages" while red dashed lines represent models with "Number of Threads" added to them. The latter performs slightly better than the former.

light, developers may exhibit distinctive dynamics of (social) activity around "reward" (*i.e.,* committership) time, *e.g.,* similar to Stack Overflow users around the time of obtaining a badge [46, 47], or IBM social networking site users around the time of levelling up [45]: substantial rise in activity before achieving the goal followed by an immediate drop.

We start by visually analysing social activity trends using a six-months window prior to and after developer initiation. After excluding developers who became committers in fewer than six months after their first interaction on the mailing list (*e.g.,* were already committers from the beginning of the project) and developers

who became committers later than six months prior to the end data collection date, for which not enough measurements of their social activity exist, our sample consists of 67 developers across the six ASF projects. For each project and periods of six months (approximated as 30 day intervals) prior to and after committership status (centred at time 0), Figure 8 depicts boxplots of the number of messages by developers in that project. Visual inspection of Figure 8 suggests a peak in social activity close to committership status (either one month before—in Axis2_c—, or one—in Ant, Lucene, or Pluto—or two—in Log4j—months after initiation), followed by a drop in communication in the following months, in all projects except Solr.

To statistically evaluate the presence of trends in the data, we split each developer's communication history into two equal-sized groups, at the initiation point, and test for the existence of increasing and decreasing trends prior to and following developer initiation. We test for a simple monotonic trend between time t and the values at time t, $y(t)$, using the non-parametric Mann-Kendall test [60]. For 3 of the 6 projects we confirm a statistically significant monotonic trend for the six months prior to developer initiation, while for one of the six we confirm only a statistically suggestive monotonic trend (Table 11). On the other hand, for the six months following developer initiation, we cannot confirm a monotonic trend in any of the projects. However, for five of the six projects (all but Solr) the test statistic (tau) is negative, indicating a downward trend.

To gain more insight into these quantitative results, we proceed to cluster the individual patterns of communication activity for the 67 developers in our sample. Since we are interested in trends or patterns of social interaction, using the observed (*i.e.,* unsmoothed) values would introduce excessive noise. Instead, we consider LOESS [61] smoothed versions with unit span of each developer's time series of social interactions, to obtain the clearest trends. Then, we perform clustering of the different communication patterns using an open card sorting approach [62], where each card contains the communication pattern of each developer. Card sorting is a technique frequently used in qualitative research for categorisation (assigning cards into meaningful groups). In case of open card sorting, there are no predefined groups, but rather groups emerge and evolve during the sorting process. To reduce bias, each of the first three authors independently assigned cards to clusters, after which review and discussion took place until all participants agreed on the final set of clusters. The results are presented in Figure 9.

We reflect on a number of observations. First of all, six out of the seven clusters (cluster 6 contains too few developers to draw any meaningful conclusions), com-

Table 11. Mann-Kendall Trend test for 6 months prior to and after becoming committer.

	6 months before		6 months after	
	tau	p-value	tau	p-value
Ant	0.20	0.00	-0.05	0.27
Axis2_c	0.11	0.18	-0.09	0.17
Log4j	0.28	0.07	-0.08	0.30
Lucene	0.21	0.00	-0.01	0.83
Pluto	0.23	0.03	-0.12	0.08
Solr	-0.01	0.90	0.01	0.89

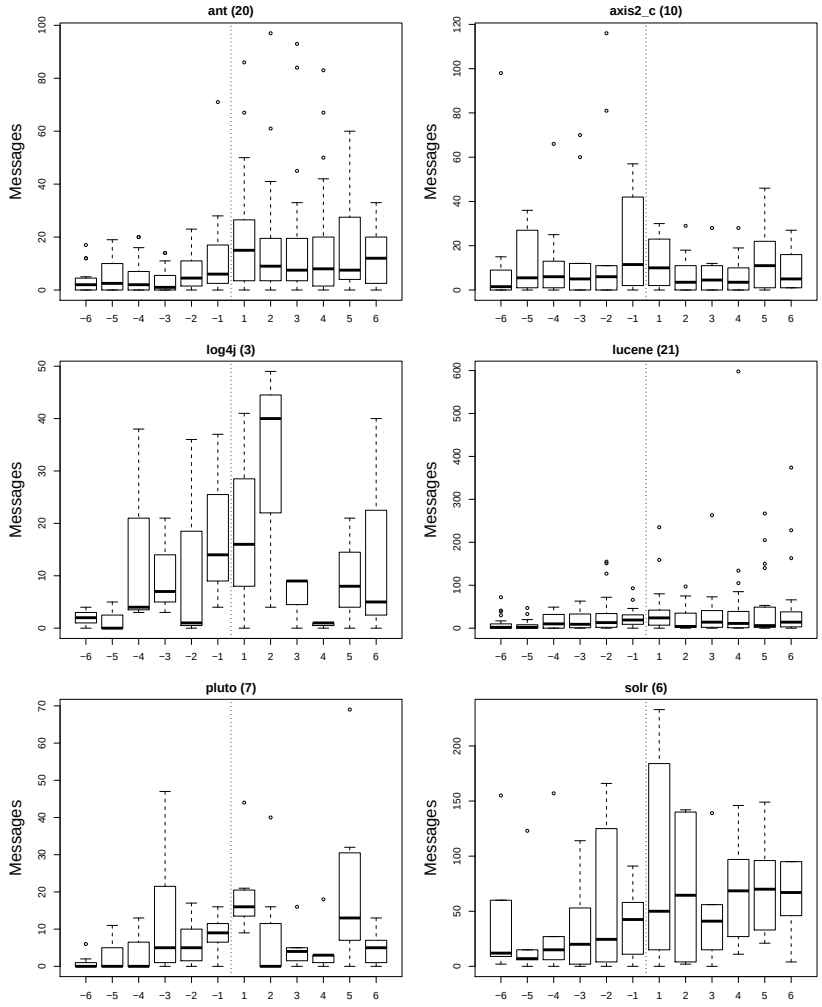

Figure 8. Boxplots of individual's communication activity around time to become a committer. The numbers in parentheses besides each project's name indicate the number of individuals in the boxplots. Developers who became committers in fewer than 6 months after their first message on the mailing list (*e.g.,* were already committers from the beginning of the project) and developers who became committers later than 6 months prior to the end data collection date, for which not enough measurements of their social activity exist, were excluded.

prising the vast majority of developers, fall into either of three groups of patterns. The first group, comprising clusters 1, 2, 3 and, to a lesser extent 5, consists of developers that exhibit the anticipated goal-directed approach behavior: a ramp-up of social activity in the period of time immediately before (cluster 1), right around (cluster 2) and shortly after (cluster 3) developer initiation, followed by a cooling off period. Developers in cluster 5 do not exhibit the drop in social activity after becoming committers.

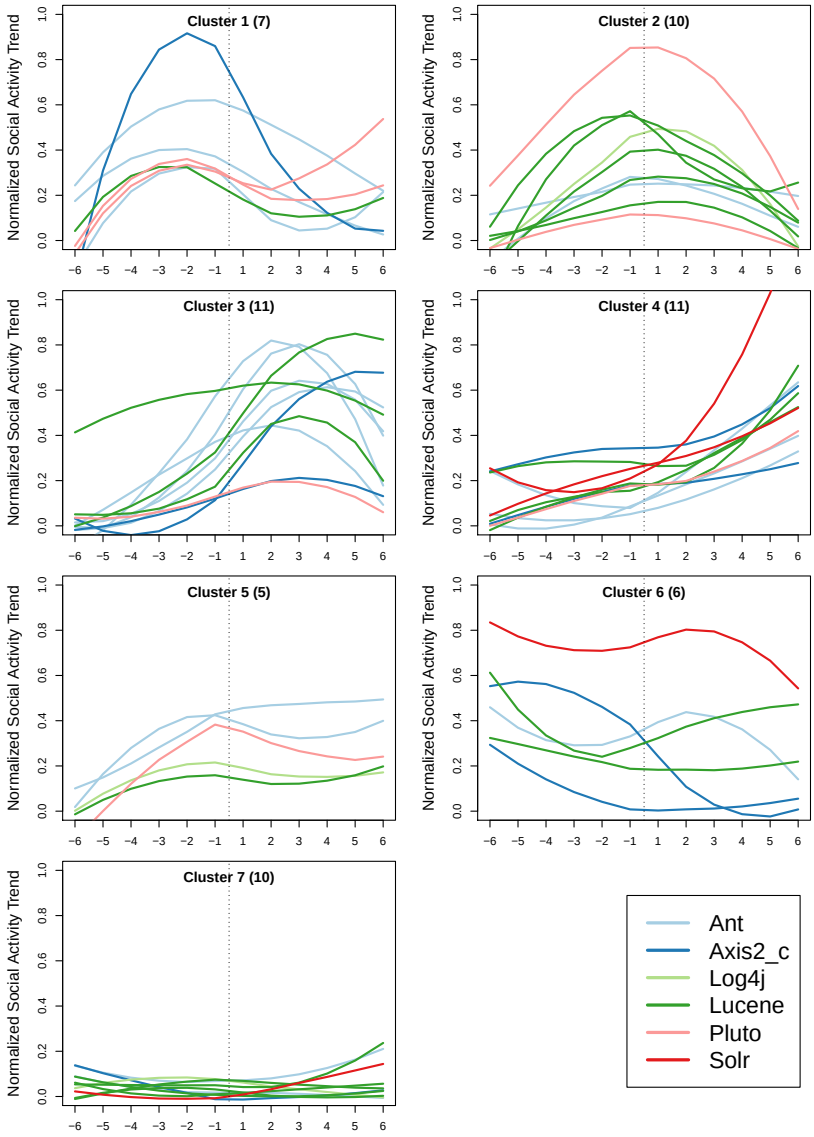

Figure 9. Patterns of social activity prior to and after developer initiation, as resulting from manual clustering based on open card sorting. Developers from the same project are assigned the same color. The numbers in parentheses besides each cluster number indicate the number of individuals in that cluster.

The second group, consisting of cluster 7, includes developers for which committership does not seem to impact their social behavior. Further manual inspection reveals that developers in this cluster often are well known members of the ASF community (*e.g.,* Apache evangelists, book authors) or committers elsewhere

already within the ASF, suggesting that established ASF contributors might need much less reputation-building than newcomers when trying to become committers.

Finally, the third group, consisting of cluster 4, comprises developers that continue to intensify their social activity beyond reaching committer status. Further manual inspection of this cluster suggests other external factors at play, such as incubator projects being "graduated" as sub-projects of Lucene (*e.g.,* Nutch) or Ant (*e.g.,* Ivy), together with the merger of their respective developer mailing lists into the mailing lists of the main projects at the time of committership.

Result 3: *Typically, a ramp-up of social activity occurs in the period of time close to developer initiation. Following a short cooling off period right after initiation, more individualized socialization patterns emerge.*

5.4 Research Question 4

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 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 12. We see that for all projects, the coefficient of "IsSecond" is negative and statistically significant across three of the six projects. In a fourth project, *solr*, the coefficient is statistically suggestive at the 10% level in both models ($p = 0.103, 0.108$, respectively). Statistically insignificance, does not imply the converse, we can only state that the result is "inconclusive". The negative sign of the coefficient, however, indicates a negative skew in the confidence intervals

Table 12. It becomes increasingly more difficult to earn trust in an OSS. Models show a dummy variable "IsSecond" which is true for individuals that $Project\ Age > median(Project\ Age)$. 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

about the predictor. In most projects "age" has a negative effect on chances of becoming a developer. Based on these observations, we conclude:

> **Result 4:** *It becomes more difficult for individuals to become committers as the project matures; late stage developers may have to put more effort to gain the same level of trust.*

6 Conclusion and Threats to Validity

We presented strong evidence for the determining role that social networking activities play in becoming a developer in the studied ASF project. Surprisingly, to this end, social communications are a better predictor than patching activity. We also present evidence that developers' early social activities in the project identify them as such. Moreover, prior to becoming developers, participants typically ramp-up their socialization activities, in anticipation of their upgraded status. 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[7], it is not hard to imagine other scenarios where developers are not chosen as we think they are.

When looking for trends in the socialization data before and after one becomes a developer, we are obviously limited by the number of committers with 12 month email history that is available in our dataset, which is 67 committers after eliminating insufficient data entries. In addition, the analysis of short time-series data is in general susceptible to noise arising from the multiple assumptions used, which in our case is mitigated to an extent by our relatively straight forward analysis.

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

[7] `http://www.apache.org/foundation/faq.html`

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.

Acknowledgements All authors gratefully acknowledge support from the Air Force Office of Scientific Research, award FA955-11-1-0246. Vasilescu gratefully acknowledges support from the Dutch Science Foundation (NWO), grant NWO 600.065.120.10N235. Part of this research was carried out during Vasilescu's visits at UC Davis.

References

1. D. M. German, "The GNOME project: a case study of open source, global software development," *Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215, 2003.
2. K. Crowston, K. Wei, J. Howison, and A. Wiggins, "Free/libre open-source software development: What we know and what we do not know," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, p. 7, 2012.
3. A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 3, pp. 309–346, 2002.
4. K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *IWPSE*. ACM, 2002, pp. 76–85.
5. Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," in *ICSE*. IEEE, 2003, pp. 419–429.
6. 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.
7. G. Robles and J. M. Gonzalez-Barahona, "Contributor turnover in libre software projects," in *Open Source Systems*. Springer, 2006, pp. 273–286.
8. N. Ducheneaut, "Socialization in an open source software community: A socio-technical analysis," *CSCW*, vol. 14, no. 4, pp. 323–368, 2005.
9. B. S. Butler, "Membership size, communication activity, and sustainability: A resource-based model of online social structures," *Information systems research*, vol. 12, no. 4, pp. 346–362, 2001.
10. B. Kogut and A. Metiu, "Open-source software development and distributed innovation," *Oxford Review of Economic Policy*, vol. 17, no. 2, pp. 248–264, 2001.
11. J. Roberts, I. Hann, and S. Slaughter, "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects," *Management science*, vol. 52, no. 7, pp. 984–999, 2006.
12. R. Fielding, "Shared leadership in the Apache project," *Communications of the ACM*, vol. 42, no. 4, pp. 42–43, 1999.
13. V. Sinha, S. Mani, and S. Sinha, "Entering the circle of trust: developer initiation as committers in open-source projects," in *MSR*. ACM, 2011, pp. 133–142.
14. C. Jensen and W. Scacchi, "Role migration and advancement processes in OSSD projects: A comparative case study," in *ICSE*. IEEE, 2007, pp. 364–374.
15. C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu, "Open borders? pImmigration in open source projects," in *MSR*. IEEE, 2007, pp. 6–6.
16. 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 *ESEM*. ACM, 2008, pp. 2–11.
17. N. Bettenburg and A. E. Hassan, "Studying the impact of social structures on software quality," in *ICPC*. IEEE, 2010, pp. 124–133.
18. C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality?an empirical case study of Windows Vista," *Communications of the ACM*, vol. 52, no. 8, pp. 85–93, 2009.
19. Y. Long and K. Siau, "Social network structures in open source software development teams," *Journal of Database Management (JDM)*, vol. 18, no. 2, pp. 25–40, 2007.
20. K. Crowston and J. Howison, "The social structure of free and open source software development," *First Monday*, vol. 10, no. 2, 2005.

21. C. De Souza, J. Froehlich, and P. Dourish, "Seeking the source: software source code as a social and technical artifact," in *SIGGROUP*. ACM, 2005, pp. 197–206.
22. E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.
23. K. Stewart and S. Gosain, "An exploratory study of ideology and trust in open source development groups," in *ICIS*. ACM, 2001, pp. 1–6.
24. 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.
25. 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.
26. G. Krogh and E. Hippel, "The promise of research on open source software," *Management Science*, vol. 52, no. 7, pp. 975–983, 2006.
27. W. Scacchi, "Free/Open source software development: Recent research results and methods," *Advances in Computers*, vol. 69, pp. 243–295, 2007.
28. C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *MSR*. ACM, 2006, pp. 137–143.
29. F. Rahman, D. Posnett, and P. Devanbu, "Recalling the imprecision of cross-project defect prediction," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 61.
30. F. Rahman, D. Posnett, I. Herraiz, and P. Devanbu, "Sample size vs. bias in defect prediction," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 147–157.
31. M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 2010, p. 9.
32. M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.
33. D. Posnett, V. Filkov, and P. Devanbu, "Ecological inference in empirical software engineering," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 362–371.
34. 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.
35. I. Herraiz, G. Robles, J. Amor, T. Romera, and J. González Barahona, "The processes of joining in global distributed software projects," in *International Workshop on Global Software Development for the Practitioner*. ACM, 2006, pp. 27–33.
36. B. Shibuya and T. Tamai, "Understanding the process of participating in open source communities," in *International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. IEEE, 2009, pp. 1–6.
37. 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.
38. M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in OSS community," in *ICSE*. IEEE, 2012, pp. 518–528.
39. R. A. Depue and P. F. Collins, "Neurobiology of the structure of personality: Dopamine, facilitation of incentive motivation, and extraversion," *Behavioral and Brain Sciences*, vol. 22, no. 03, pp. 491–517, 1999.
40. W. Schultz, "Behavioral theories and the neurophysiology of reward," *Annu. Rev. Psychol.*, vol. 57, pp. 87–115, 2006.
41. R. E. Lucas, E. Diener, A. Grob, E. M. Suh, and L. Shao, "Cross-cultural evidence for the fundamental features of extraversion." *Journal of personality and social psychology*, vol. 79, no. 3, p. 452, 2000.
42. M. C. Ashton, K. Lee, and S. V. Paunonen, "What is the central feature of extraversion? social attention versus reward sensitivity." *Journal of personality and social psychology*, vol. 83, no. 1, p. 245, 2002.
43. S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *CHI*. ACM, 2011, pp. 2425–2428.
44. R. Cheng and J. Vassileva, "Design and evaluation of an adaptive incentive mechanism for sustained educational online communities," *User Modeling and User-Adapted Interaction*, vol. 16, no. 3-4, pp. 321–348, 2006.

45. R. Farzan, J. M. DiMicco, D. R. Millen, C. Dugan, W. Geyer, and E. A. Brownholtz, "Results from deploying a participation incentive mechanism within the enterprise," in *CHI*. ACM, 2008, pp. 563–572.
46. A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, "Steering user behavior with badges," in *WWW*. ACM, 2013, pp. 95–106.
47. S. Grant and B. Betts, "Encouraging user behaviour with achievements: an empirical study," in *MSR*. IEEE, 2013, pp. 65–68.
48. A. Begel and B. Simon, "Novice software developers, all over again," in *Proceedings of the Fourth international Workshop on Computing Education Research*. ACM, 2008, pp. 3–14.
49. G. Dai and K. P. De Meuse, "A review of onboarding literature," *Lominger Limited, Inc., a subsidiary of Korn/Ferry International*, 2007.
50. T. N. Bauer and B. Erdogan, "Organizational socialization: The effective onboarding of new employees." 2011.
51. T. N. Bauer, T. Bodner, B. Erdogan, D. M. Truxillo, and J. S. Tucker, "Newcomer adjustment during organizational socialization: a meta-analytic review of antecedents, outcomes, and methods." *Journal of applied psychology*, vol. 92, no. 3, p. 707, 2007.
52. B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens, "On the variation and specialisation of workload—a case study of the GNOME ecosystem community," *Empirical Software Engineering*, pp. 1–54, 2013.
53. 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, 2012, pp. 592–595.
54. M. Goeminne and T. Mens, "A comparison of identity merge algorithms for software repositories," *Science of Computer Programming*, vol. 78, no. 8, pp. 971–986, 2013.
55. A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. v. Deursen, "Communication in open source software development mailing lists," in *MSR*. IEEE, 2013, pp. 277–286.
56. N. Bettenburg, E. Shihab, and A. E. Hassan, "An empirical study on the risks of using off-the-shelf techniques for processing mailing list data," in *ICSM*. IEEE, 2009, pp. 539–542.
57. B. Vasilescu, A. Serebrenik, P. T. Devanbu, and V. Filkov, "How social Q&A sites are changing knowledge sharing in open source software communities," in *CSCW*. ACM, 2014, pp. 342–354.
58. J. Cohen, *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.
59. Q. Vuong, "Likelihood ratio tests for model selection and non-nested hypotheses," *Econometrica: Journal of the Econometric Society*, pp. 307–333, 1989.
60. H. B. Mann, "Nonparametric tests against trend," *Econometrica: Journal of the Econometric Society*, pp. 245–259, 1945.
61. W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *Journal of the American statistical association*, vol. 74, no. 368, pp. 829–836, 1979.
62. D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.