Does the Initial Environment Impact the Future of Developers?

Minghui Zhou
School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, China
zhmh@pku.edu.cn

Audris Mockus
Avaya Labs Research
233 Mt Airy Rd, Basking Ridge, NJ
audris@avaya.com

ABSTRACT

Software developers need to develop technical and social skills to be successful in large projects. We model the relative sociality of a developer as a ratio between the size of her communication network and the number of tasks she participates in. We obtain both measures from the problem tracking systems. We use her workflow peer network to represent her social learning, and the issues she has worked on to represent her technical learning. Using three open source and three traditional projects we investigate how the project environment reflected by the sociality measure at the time a developer joins, affects her future participation. We find: a) the probability that a new developer will become one of long-term and productive developers is highest when the project sociality is low; b) times of high sociality are associated with a higher intensity of new contributors joining the project; c) there are significant differences between the social learning trajectories of the developers who join in low and in high sociality environments; d) the open source and commercial projects exhibit different nature in the relationship between developer's tenure and the project's environment at the time she joins. These findings point out the importance of the initial environment in determining the future of the developers and may lead to better training and learning strategies in software organizations.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*process metrics*; D.2.9 [**Software Engineering**]: Management—*productivity*

General Terms

Measurement, Performance, Human Factors

Keywords

Socio-technical balance, initial environment, relative sociality, learning trajectory

1. INTRODUCTION

The most critical tasks in software projects require "expertise across multiple areas", however, "there are few staff to choose from" according to an expert developer [1]. One possibility suggested by a software project manager, is that many developers tend to focus on the modules they are familiar with, and rarely communicate outside their narrow circle of colleagues to gain expertise in other areas. The software engineering literature has investigated the important role of social and communication aspects in a developer's work. They might impact developer productivity (Cataldo et al [2]) and they might affect software quality (Cataldo et al [3]). Furthermore, cognitive scientists have argued that interacting with partners is significantly better than learning alone [5]. In other words, the developers need both technical and social skills to be capable of solving critical tasks, though that might present two contradicting or at least competing learning goals.

On the other hand, there may be obstacles for the developers to achieve socio-technical balance, even when they have a strong motivation to cultivate their social and technical trajectories, because the project environment, in particular, the environment at the time a developers joins (i.e., the initial environment for the developer), may have a significant impact on the individual. For example, in many offshoring projects, the developers in the offshore location were considered to be incompetent to implement new feature development in legacy projects: "I don't know if people are "climbing up" (moving from defect fixing to new development) in this site," because "initially nobody could get trained by experienced mentors", according to an outsourcing manager. Therefore, "the offshore team really needs time working with onshore developers to gain mature practices," according to the same manager.

This anecdotal evidence sparked our interest to investigate how the initial environment may impact the developers' learning trajectories, in particular, the achievement of social and technical balance. Improving this process may help understand how to increase the number of developers capable of solving critical tasks, to improve the developers' training, and to facilitate the project's success.

We have to overcome two challenges to proceed with this investigation. First, we need to measure the socio-technical balance, second, we need to determine how the initial environment affects the trajectories of developers. In addition to the challenges of measuring the social and technical achievement in general, we also need to derive these measures from commonly available project data, such as version control system and problem tracking system. Such data are difficult to obtain and even more difficult to interpret. For example, Cataldo et al. [4] compared an MR-induced logical dependency graph on source code files with a graph induced by instant

[1]The quotes, including the latter ones, are obtained from the interviews conducted in our former work [20].

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

messaging communications among developers working on the corresponding files. The lack of communications where a technical dependency exists was assumed to indicate a potential problem. Unfortunately, there are many other forms of communications among the developers beyond what the instant messaging could capture.

In this study we don't look for, or study the mismatches between the technical dependencies and the communication paths, instead we focus on the individual developers by comparing a developer's achievement from the social, and the technical, aspects. In particular, we propose a measure of Relative Sociality (RS) to observe the developer's socio-technical balance: the ratio of the size of a developer's communication network to the number of tasks she participates in. The former is modeled by the number of people the developer interacted with in her task workflow. The latter is represented by MRs in problem tracking system. In general, the number of people a developer encounters in the MR workflows would strongly depend on the number of MRs she participates in. This number of MRs may vary substantially over time because of various envirnmental influences, however, the ratio in RS adjusts for such influences by normalizing the social network using the number of tasks (MRs).

We propose several hypotheses about how the initial project environment affects the developer's RS trajectory. We consider how the project's RS (obtained from RSes of project participants) at the time the developer joins the project affects her trajectory. We investigate three open source projects and three traditional (or commercial) projects and partition each project's time-line into low, high, and medium RS periods as described in Section 4.4. We find that the developers' RS trajectories depend on the type of interval when she joined the project, that the project's RS measure at the time a developer joined affects the probability that a developer would become one of the long-term (28 or more months) and productive (top 90%) developers (we refer to such developers as Long Term Contributors or LTCs), and that a project environment when RS measure is high is associated with a higher intensity of new contributors. Moreover, the open source and commercial projects show their different nature through this measure, in particular, when the project's RS is low, the intensity of the joiners who eventually become LTCs is high in open source projects but that is not the case for traditional projects.

The main contributions of this work include: a) to propose and evaluate the RS measure of socio-technical balance, b) to propose measures of the project's success (the intensity of joiners and the probability for a joiner to become an LTC), c) to demonstrate the associations between the project environment, these measures of success, and developers' learning trajectories. In particular, we quantify the developers' learning trajectories, and establish the association between the project environment (reflected through RS measure) and the intensity of new contributors or the chances that a new contributor would become an LTC.

We review the related work in Section 2, introduce the projects and the methodology in Section 3 and Section 4. Section 4.4 discusses the design of RS measure, Section 5 lists the hypotheses, and Section 6 reports the results. We discuss the limitations of this work in Section 7 and conclude in Section 8.

2. RELATED WORK

There is a substantial amount of work in software engineering literature that relates to this study and we roughly classify it into four groups: investigating the nature of communications, determining the impact the communications have on developers' work, studying the influence of the environment on developers and their work, and proposing the best practices of socio-technical congruence for software development.

The first group of work tries to discover the nature of communications in a developer's programming life. For example, communication plays an important role for developers as Begel et al. [1] found in Microsoft. Majority of indicated developer needs involved discovering, meeting, and keeping track of people, not just code. Communication should not be regarded as something to be increased or limited by itself, what matters is the quality of communication experience, as Nakakoji et al. highlighted in [14].

A traditional approach to investigate socio-technical congruence is measuring the overlap between programming dependencies and communication paths. For example, this approach has been used to illustrate how the mismatch between social (primarily the communications among developers) and technical aspects in developer work has an impact on project schedule, software quality, and developer effort [4, 2, 3]. The basic idea is that the lack of communication where a technical dependency exists may indicate a potential problem.

A good example of the environmental influences is Conway's Law [6] stating that the structure of an organization is reflected in the structure of the systems it designs. The environment is represented by the organization's structure, and it strongly influences the way people produce the system. Mockus [11] investigated the project environment measured via organizational change and the intensity of staff reductions, and found that the proximity to such environmental change was associated with reductions in software quality. Other environmental factors, for example, the accessibility of resources provided by the project, were found to affect the developers' training and learning [21].

Regarding best practices, Cataldo et al. [4] described an example of improved quality of communications where the most productive developers changed their use of electronic communication media over time, achieving higher congruence between coordination requirements and coordination activities. New models and approaches are proposed because "existing software engineering tools ... separate time and tasks in concrete but isolated process steps" [16]. For example, Redmiles et al. [16] proposed a way to support collaborative work called Continuous Coordination. It stresses the importance of integrating the coordination activities within the programming environment, and of making developers aware of the need to communicate. It simultaneously minimizes the distraction of software developers by using formal configuration management mechanisms and informal visual notification and awareness techniques. Ye et al. [19] proposed a Socio-Technical Framework to provide incremental support of information acquisition from immediately relevant information, to contextual information, and to relevant peers. It guides the design of systems that support information seeking during various phases of software development.

As described above, many studies attempt to systematically treat both technical issues and social issues in a programming context. However, the quantification of how the two aspects interact in practice, of the relationship between a developer's social and technical skills and, in particular, of how the initial environment affects the developer's socio-technical balance, has not been investigated.

3. CONTEXT

We have investigated both open source and commercial projects presented in Table 1. Two open source projects implement middleware functionality (JBossAS and JOnAS) and one implements user interface functionality (Mozilla). The commercial projects were produced by a large US-headquartered corporation, belong to the

telephony domain, and provide various functionalities of a telephone switch and messaging system and are referred to as D, F, and K. All three commercial projects and Mozilla have more than 10 years of history.

Table 1: Projects

Project	Years	MLOC[2]	Domain	Cntrbtrs
JOnAS	8	0.2	Middleware	279
JBossAS	5	0.3	Middleware	2516
Mozilla	12	1.5	UI	62056
D	> 20	5	Telephony	1966
F	> 10	3	Embedded telephony	464
K	> 15	1.5	Messaging	223

4. METHODOLOGY

We obtained qualitative information about the projects through extensive past and ongoing interactions and studies. For example, one of the authors had lead a team that contributed a substantial portion of JOnAS software and continues to contribute to the project. Mozilla was studied by another author in the past [12]. We rely on methods described by Mockus [10] to conduct our quantitative study. We conduct the steps outlined in the following subsections to obtain sufficient quality data.

4.1 Retrieve Raw Data

For the open source projects, we wrote crawlers to obtain the modification request (MR) histories and details from their web pages. JOnAS and JBossAS use JIRA for problem tracking, though they used other tracking systems in the past. JBossAS used Tracker on SourceForge prior to 2006 and migrated to JIRA all issues raised since 2004. JOnAS used GForge before October, 2009 and migrated to JIRA but hasn't transferred the issues from GForge. Mozilla uses Bugzilla to track issues. We obtained information for all issues in XML format as well as the activity history for each project from all the sources in February, 2010. For Mozilla we also used the issues we obtained in 2001 to validate the data obtained in 2010 (which included these older issues as well). For the commercial project F, we obtained the MR history data recorded in the MR history table of ClearQuest. For projects D and K, we obtained Sablime MR history files.

Independent of which system an MR is retrieved from, each MR has a history, from the time somebody reported it until the time somebody closed it (it also may remain open at the time of the study). During that period a sequence of events take place: MR is created, assigned, submitted, tested, and closed. It may also be reassigned, its attributes changed, comments, debugging traces, etc added. Each such event has an associate date, time, the type of action, and the developer performing the action. The transition from one action to another may involve a change of actors and such work hand-offs we consider to be edges in the project workflow graphs (see Appendix for precise definitions). In this study every observation is an edge in such workflow graph. More specifically, we obtained the chronological sequence of tuples containing: date and time, MR ID, contributor ID, action, and value. We then created a link between every pair of contributors immediately adjacent in this sequence. The assumption is that such transfer of MR ownership indicates a likely communication between these contributors. Even in cases where a verbal or other type of communication does not accompany such a hand-off, it still represents an artifact-mediated communication as demonstrated by, for example, [17].

We considered the first appearance of a contributor in this graph as the date the contributor joins the project. Given the minimum of 28 months of tenure (the time a developer stays with the project) for the selected sample of contributors, we considered it to be a reasonable approximation even though the actual participation had to start before that time to allow for time needed to at least learn how to use the MR system. Regarding the minimum tenure of LTCs, we want it to be as long as possible, because it takes at least three years to become fluent in a large scale project [20], but, at the same time, the sample for JOnAS was too small if we dropped everybody who joined within the last three years. In general, the definition of LTC should probably be based on the time it takes to become fluent in the project [20].

4.2 Clean Raw Data

As described above, several projects have changed problem tracking systems, potentially introducing data artifacts.

JBossAS data spanned 10 year period. However, the data until January 2005 had only 12 developers (with no one joining or leaving) from 2001 to 2004. Therefore we did not consider this period because it is likely to have data issues and is not suitable to observe newcomers.

For Mozilla, we compared the 2010 extract with the 2001 extract and found some differences. In particular, 2010 extracts had an option to obtain MR details in XML format, which allowed us to associate contributor IDs with their full names. 2001 extracts were only in HTML and some of the contributor IDs have changed over time. For example, "buhr+mozilla2" has changed to "buhr+mozilla." The 2010 extract for older bugs used the ID present in the 2001 extract, however the IDs representing full email address were stripped of the domain part (e.g., mcafee@netscape.com changed to mcafee), presumably to thwart spam-bot crawlers.

For all projects we paid a particular attention to identifying instances when a contributor ID has changed over time. In traditional projects we used multi-year extracts of the organization's reporting structure to identify the mapping between people and developer IDs. For open source projects we used the mapping between the ID and the name available in XML extracts and, for JBossAS we also used mailing lists. We used various heuristics to identify multiple IDs for the same person. In particular, IDs corresponding to the same name (or similarly spelled name) were selected for manual verification if the joining date of one ID followed the last contribution date of another ID associated with the same name. For example, in JBossAS developer with the ID asaldhana changed his or her ID into anil.saldhana after 2006, and that appears to be a common pattern, i.e., the period before 2006 the ID was constructed by extracting the first character of the first name and appending the surname, but after the migration to JIRA, the full first name separated by a period from the surname were commonly used as the new ID.

For Mozilla and projects D, F, and K we identified a number of logins that were generic or administrative ones because we focused on the behavior of ordinary contributors. Examples of such logins were: "webmaster", "cqadmin", "mozilla", "bugzilla*" and so on. To facilitate such identification in Mozilla we looked for instances where the same ID was associated with multiple names. Multiple spellings of a name are not uncommon (e.g., with or without the middle name), but some IDs had tens or more associated names (e.g., "bugzilla") suggesting that it was used for administrative purposes and not suitable to measure individual developer's behavior.

4.3 Determine the Questions to Be Answered

In this study we tried to answer the question: does the project environment at the time a developer joins affects her future trajectory of relative sociality? Thus we need to measure the developer's technical achievement and social activities in a software project,

how they represent her (and the project's) relative sociality, and how the project's relative sociality affects the trajectory of developer's relative sociality.

4.4 Measure the Socio-Technical Balance

Using data from six projects, we model the social learning [3] using the size of the social network ($NumLogins$) a developer accumulated in the course of resolving/assigning MRs over a particular time period, i.e., a month in this study. According to Schmidt [17] an MR system may be considered as a Social Mechanism of Interaction supporting the articulation of the distributed activities of multiple actors with respect to the system and the field of work it represents. In other words, MRs represent the monitoring, handling, in particular, coordination of the tasks, and a software project proceeds by creating and resolving MRs. Therefore, we calculate the social network of a developer by adding up all the working relationships (immediately adjacent hand-offs in the MR workfows) she was involved in during the considered time period. For example, Developer Alice has been involved in two MRs during a specific month: in one MR she received work from Developers Tom and Jerry, and in another MR from Developers Kiki and Jerry. Therefore the social network of Developer Alice is Tom, Kiki, and Jerry, and its size $NumLogins$ is three.

We measure the technical learning by the developer experience, in particular, by the number of MRs ($NumMRs$) the developer participated in during a month, see, e.g, Zhou and Mockus [20]. Because Developer Alice has been involved in two MRs, her technical achievement in that month is $NumMRs = 2$.

We take the ratio of the size of a developer's communication network to the number of tasks she participates in to measure the balance between her social learning and technical learning, and we call it the Relative Sociality (RS). The periods with high RS would indicate that relatively more social learning is taking place, while the periods with low RS are indicators of more technical learning.

In addition to the developer RS, we also define the RS for an entire project during a month as the geometric average of the RSes of individual developers active in the project during that period. We use it to measure the social environment of the project at that time. In particular, the periods where the RS measure is high are supposed to indicate that the project environment is more social. We partition each project's time-line into three types of periods: the 25% of the time when the project's RS has the lowest values (low level), 25% of the time it has the highest values (high level), and the remaining 50% of the time it has values in between (medium level). We then classify the developers according to the type of period at the time they joined the project. Figure 1 illustrates the project's RS measure (represented by Y-axis), the corresponding periods (represented by the two horizontal lines showing the boundary between low, medium, and high values of RS), and the developer joining times (represented by L/H/M symbols) for Project F. More generally, we want to investigate the aspects of RS trajectory that are relevant to development practice. The developer tenure with the project is crucial for commercial projects because it takes a relatively long time for developers to become productive (see, e.g., [13]) and even more time to be able to do central tasks (see, e.g. [20]), e.g., mentoring newcomers. The ability to train and retain productive developers can, therefore, be considered as a measure of project's success. For similar reasons the developer tenure may be important in open source projects. Also, the LTCs

[3]Our approach to define learning closely follows the Legitimate Peripheral participation approach [9] that emphasizes the role of practice in the learning process. Therefore, instead of the term learning we could have used the term experience throughout the paper.

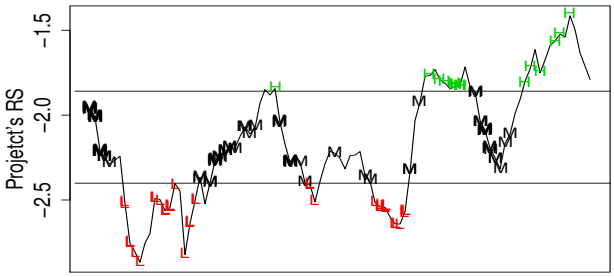

Figure 1: RS for Project F.

may provide additional stability and increase the chances that an open source project is successful. Accordingly, we measure how the social environment of a project when a developer joins impacts her becoming an LTC in the future, to understand how a software project attracts LTCs and thus becomes successful.

It is important to note that the amount of noise in the RS measure depends on the length of interval over which it is computed. For the data reported in this paper we compute one value for each month, but in order to reduce the amount of noise we take into account the activity over the past six months to obtain the RS value. In other words, the numerator is the size of the workflow network over past six months and the denominator is the number of MRs encountered over the same period. We experimented with various window sizes from one to twelve months before choosing the six month window (more discussion is in Section 7).

Also, the reported data considers developer's social network based on the number of developers from whom she has received the work. Thus, it does not include individuals that follow her in the workflow graph. This choice was based on the finding by Herbsleb and Mockus [8] that the size of the inflow network significantly decreased developer productivity. In practice, the reported results do not change notably if we expand the social network by including developers subsequent in the workflow graph.

5. HYPOTHESES

Our primary focus is to investigate how the project environment at the time a developer joins may impact the evolution of her socio-technical balance.

Initial environmental conditions may substantially influence the technological trajectories of business firms, see, e.g., Zyglidopoulos [22]. Four mechanisms through which this influence may be carried out are technological paradigm, dominant logic, organizational structure, and configuration. These mechanisms act as carriers of initial influences and constrain the future technological developments, thus restricting the possible technological trajectories a firm can follow [22]. A firm is organized through individuals, and the existence of a firm is for the sake of individuals, thus it's of great interest to look at the technological trajectories at the individual level. Such investigation may, in turn, reflect the success of an organization, or at a lower level, of a software project. We presume that the environment of a project when a developer joins has a big impact on her future trajectory in the project, for example, Developer Kiki becomes one of the most productive developers later on because she has been mentored by the expert developers when she joined. Therefore:

Hypothesis 1. *The social environment of the project when a developer joins affects her future trajectory.*

As discussed in Section 4.4, the contributors who stay with the project for a long time and are productive (LTCs) are important for a project's success, therefore we provide more specific instances of Hypothesis 1 focusing on the association between software project's RS and the emergence of such important contributors.

During the time periods when the project sociality is low, the developers who join may receive more support and attention from the existing project members (because, presumably, they have more time to spend on mentoring activities, and, in OSS projects, more appreciation of external contributions), thus it is more likely that they would stay with the project for a long time and be productive. Therefore, a more specific instance of Hypothesis 1 can be phrased as:

Hypothesis 2. *Low values of project sociality measure at the time a developer joins increases the probability that this developer will become one of long-term contributors.*

New people joining the project are crucial for a project's survival, because the existing contributors eventually leave or die, or may also run out of new ideas. The increase in the project's RS measure is likely to reflect an increase in activities that involve active development or external events that may raise the profile of the project and provide opportunities that attract new developers. Let's call the number of new developers joining per unit time to be intensity of joining developers. We expect that an active project would have a high intensity of joiners:

Hypothesis 3. *The intensity of developers joining a project is high when the project's sociality is high.*

As stated in Hypothesis 2, the low project's RS may increase the chances that a developer will become an LTC, but the number of joiners may be too low (according to Hypothesis 3) to generate a meaningful number of LTCs or it may be high enough to generate more LTCs than periods of medium or high RS. In an open source project the periods of low RS may indicate times of lower project popularity when the joiners are more likely to have a more genuine (not popularity driven) interest in the project and may also have more opportunities to get deeply involved, for example, by being able to add major functionality. Both of these conditions may increase the probability of becoming an LTC. In traditional projects the periods of low RS may reflect external business conditions and thus associated with lower hiring activity. However, it is not clear why these two potential avenues to become an LTC in open source project would also be available for joiners of traditional projects. Therefore:

Hypothesis 4. *The intensity of long-term contributors joining an open source project is high when the project sociality measure is low.*

While Hypothesis 2 considers the probability for a new contributor to become an LTC, Hypothesis 4 considers the intensity of incoming LTCs. It is also of interest to observe if there are differences among LTCs depending on when they joined the project. A lot of attention has been devoted to the differences between good and bad programmers, for example, Curtis observed an order of magnitude difference in developers' productivity [7], but how the productive programmers might differ has not been investigated. Software projects require people with different types of expertise, thus determining what factors affect the type and the extent of that expertise may help projects train developers with the most relevant skills. We expect that the magnitude of project's RS at the time the LTCs join may affect their future trajectories.

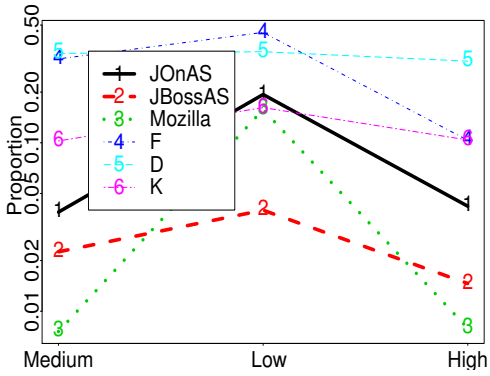

Figure 2: Proportion of LTCs

Hypothesis 5. *The LTCs joining at different levels of project's RS follow distinct RS trajectories.*

6. RESULTS

Hypothesis 1 will be considered by investigating more specific Hypotheses 2, 3, 4, and 5.

To test Hypothesis 2, we look at the proportion of the developers joining during Low, Medium, and High periods who would eventually become LTCs (staying with the project for at least 28 months and whose productivity expressed as a ratio $\frac{MRs}{Tenure}$ exceeds the 10th percentile among developers with tenure of 28 months or more). Table 2 lists the numbers of joiners who became LTCs and who did not become an LTC. Figure 2 shows the proportion of developers who became LTCs joining at different periods for each of the considered projects, in which X-axis represents the three time periods and Y-axis represents the proportion of LTCs. From Figure 2 we can see that when the project's RS measure is low, the joining developers are more likely to become LTCs. For example, in Mozilla, 1998 developers joined during 25% of the time when the project's RS had the lowest values, and 315 or about 16% of them grew into LTCs, the highest proportion among the three initial conditions.

In Figure 2 the thick lines indicate open source and thin lines commercial projects. We can see that a much higher proportion of joiners become LTCs in traditional projects (from 10% to 50%) than in open source projects (from 1% to 20%), perhaps reflecting the fact that the joiners are screened (through a hiring or an assignment process) in traditional projects but not in open source projects. It may also indicate that the motivation other than project participation (e.g., benefits of job) may reduce the probability of leaving even in an adverse environment. To answer our hypothesis the cap

Table 2: Joiners and LTCs.

Prj.	Low		Mid		High	
	not	LTC	not	LTC	not	LTC
JOnAS	25	6	124	5	114	5
JBossAS	362	15	781	18	1282	19
Mozilla	1683	315	41897	320	17700	141
F	46	38	161	74	130	15
D	280	149	702	360	330	145
K	57	11	70	8	69	8

($\cap$) pattern in Figure 2 indicates that for all projects the probability that a newcomer will become an LTC is highest during the periods when the project's RS is low. To confirm this pattern we fitted the following logistic regression [4] model:

$$L = Initial + Type + Project \quad (1)$$

[4]Logistic regression is a common way to model proportions where the proportion is related to predictors via a logistic function.

where L is an indicator of the long-term contributor (LTC), $Initial$ is the factor indicating the project's RS level at the joining time ("Medium", "Low", and "High"), $Type$ is an indicator of a traditional (non-OSS) project, and $Project$ is the project indicator. We used *glm* function with *binomial* family from R system [15] to obtain the estimates in Table 3. In general, an N-level categorical variable in a regression is represented by N-1 indicator-functions plus an intercept for the default level. In this model, the coefficients for the "Low" and "High" are contrasts with the "Medium" level, the coefficients for the project indicators are contrasts with Mozilla. Therefore the intercept represents "Medium"-RS for Mozilla. For example, "Medium"-RS for project K can be obtained by adding the coefficient for "Trad." to the intercept. We can see that while the high values of RS at the time of joining lead to approximately the same probability to become an LTC as the medium values (the coefficient of $-.1$ and not significant), the low RS values are associated with a significantly higher predicted probability that a developer would become an LTC. For example, it translates into a $(e^{4.5-0}+1)/(e^{4.5-1.89}+1) = 5.6$ times increase for Mozilla. Also, as noted above, the traditional projects have a significantly higher predicted probability that a developer would become an LTC, thus confirming the trends visible in Figure 2. To interpret the fitted coefficients for the initial conditions,

Table 3: Probability of becoming an LTC. Deviance explained: 0.88

	Estimate	Std. Error	Pr(>|z|)
(Intercept)	-4.50	0.04	0.00
Low	1.89	0.07	0.00
High	-0.10	0.07	0.16
Trad.	1.66	0.22	0.00
D	1.70	0.22	0.00
F	1.47	0.25	0.00
JBossAS	0.10	0.15	0.52
JOnAS	1.31	0.27	0.00

the predicted probability [5] to become an LTC for Project D goes from 22% for joiners at high RS to 68% for joiners at low RS or almost three times higher. For Mozilla, the high RS corresponds to predicted probability of 1% and low RS to 7%, or seven times higher! This demonstrates that the conditions at the time of joining expressed as the project's RS are associated with dramatic changes in the predicted probability that a joiner would become an LTC thus supporting Hypothesis 2.

To investigate Hypothesis 3 we look at the intensity of developers joining a project during the three periods introduced above. Table 2 shows that when the project's RS is high, the intensity of developers joining the project is also high. In order to enable visual comparisons among the projects, we normalize the joining intensity by the total number of project joiners, as shown in Figure 3. For example, in Mozilla, $17700+141$ developers (non-LTCs plus LTCs, see Table 2) joined when the project's RS was high and $1683+315$ joined when the project's RS was low. The corresponding intensities of joining developers are $\frac{17700+141}{0.25}$ (the number of joiners divided by time units) and $\frac{1683+315}{0.25}$ which are also used in the model specified by Equation 2. Figure 3 normalizes these numbers and for the high RS it shows $\frac{17700+141}{0.25*62056}$ (62056 is the total number of project joiners) while for the low RS it shows $\frac{1683+315}{0.25*62056}$. Figure 3 shows that, generally, low project's RS is associated with low intensity

[5]The predicted probability could be obtained through applying the inverse logistic function on the regression formula 1, i.e., p=1/(exp(-L)+1)

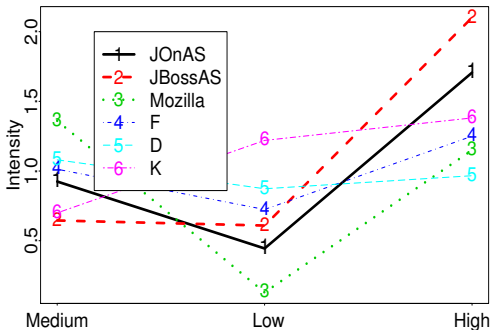

Figure 3: Intensity of new contributors.

of joiners supporting Hypothesis 3. One exception is Project K, in which the intensity of joiners is higher during low RS period than during medium RS period. Also, the traditional projects appear to have less variation in the joining intensity among different levels of RS as compared to open source projects.

The estimates for the generalized linear model for the overdispersed Poisson distribution[6] for the intensities are shown in Table 4. The model is:

$$Intensity = Initial + Project \quad (2)$$

where $Intensity$ is the joining intensity for a specific project at its specific RS time, $Initial$ is the factor indicating the project's RS level (the same as in Equation 1), and $Project$ is the project indicator. In contrast to Equation 1, we omit a separate predictor for traditional projects. The intercept, therefore, represents Project K for "Medium"-RS. We used *glm* function with *quasipoisson* family from R system [15] to obtain the estimates. Only coefficients for "low"-RS and for Mozilla are statistically significant. The results are consistent with Figure 3 and show that the joining intensity is significantly lower when the project's RS is low.

Table 4: Intensity of new contributors. Deviance explained: 0.98

	Estimate	Std. Error	Pr(>|t|)
(Intercept)	5.89	1.27	0.00
Low	-2.01	0.34	0.00
High	-0.11	0.17	0.53
D	2.05	1.35	0.16
F	0.63	1.57	0.70
JBossAS	2.42	1.32	0.10
JOnAS	0.15	1.73	0.93
Mozilla	5.41	1.27	0.00

In Hypothesis 3 we look at the intensity of all contributors joining a project at different periods. In contrast, Hypothesis 4 considers the intensity of LTCs joining a project. Therefore we calculated the intensity of joiners who eventually became LTCs with the approach we used for the intensity of joiners in Figure 3. Figure 4 shows that when the project's RS is low, the intensity of LTCs joining the project is highest for open source projects, but not for traditional projects. To confirm this pattern, we used Equation 2 for the

[6]A regular Poisson distribution results in highly significant coefficients for all of the predictors, but it is reasonable to assume that the newcomer counts may be overdispersed Poisson (variance being larger than the mean).

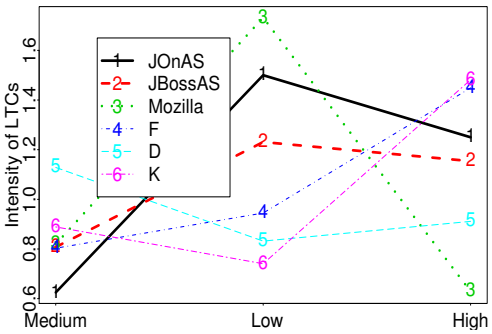

Figure 4: Intensity of new LTCs.

model ,but instead of counting all new contributors in the response $Intensity$ we counted only the new LTCs joining the project. The fitted coefficients for the model of three open source projects are shown in Table 5 (the intercept here represents "Medium"-RS for JBossAS) and support that pattern and Hypothesis 4, i.e., the LTC joining intensity is highest for low values of RS in open source projects.

Table 5: Intensity of new LTCs. Deviance explained: 0.98

	Estimate	Std. Error	Pr(>|t|)
(Intercept)	3.78	0.28	0.00
Low	0.67	0.16	0.01
High	-0.04	0.18	0.84
JOnAS	-1.16	0.52	0.09
Mozilla	2.66	0.26	0.00

To investigate Hypothesis 5 we fit the learning trajectory of a developer over her tenure as a generalized additive mixed effects model *gamm* [18] shown in Equation 3. The model fits a separate curve for LTCs who joined at each level of project's RS: high - H/low - L/medium - M.

$$RS = S_1(T|M) + S_2(T|L) + S_3(T|H) + Dvlpr. \quad (3)$$

The response is the developer's RS and the predictors include her tenure T, the level of the project's RS when she joined (L, M, or H), and the developer's ID. We consider the developer's ID as a random effect. $S_i(i = 1, 2, 3)$ are the smoothing functions such as splines that allow fitting a regression to a curve of unknown shape. The basic idea of *gamm* is to fit an unknown shape with a minimal number of parameters. Thus, a tradeoff between the best fit and fewest parameters (smoothness) is obtained. If the observed data can be reasonably explained through a linear relationship between the predictors and the response, the fitted smoothing function will simply be a straight line making it equivalent to a multiple regression. A variety of methods can be used to attain a balance between the best fit and smoothness. We used default parameters of function *gamm* in R package *mgcv* [15].

The resulting fit for the open source projects is shown in Figure 5 and for traditional projects in Figure 6, in which X-axis represents the developers' tenure and Y-axis represents the fitted RS trajectory.

Figures 5 and 6 show that the LTCs who joined the project at more social times (right column, i.e., the project's RS is high) tend to decrease their social learning relative to technical learning over time (RS goes down). In particular, JOnAS, JBossAS, Mozilla, and K show a clear downward trend while D mostly goes down and F increases. The trajectories of individual LTCs who joined during periods of low RS increase for JOnAS and K, increase after an early dip for Mozilla, F, and D, and fluctuate for JBossAS. While it is possible to argue about the causes and extent of these differences among trajectories of developers starting in different environments, here we focus on confirmatory analysis establishing their presence. To accomplish that we compare the model in Equation 3 that fits a separate curve for each level of project's RS to a simpler model where a single curve is fit for all developers, therefore the predictors only include the developer's tenure T and her ID $Dvlpr$:

$$RS = S(T) + Dvlpr \quad (4)$$

Table 6: ANOVA for trajectories for the three initial conditions.

Project	Model	DF	logLik	L.Ratio	p-value
JOnAS	1	11	-272.5633		
JOnAS	2	7	-286.8718	28.62	$< .0001$
JBossAS	1	11	-1196.254		
JBossAS	2	7	-1205.078	17.65	0.0014
F	1	11	-4961.845		
F	2	7	-4994.775	65.86	$< .0001$
Mozilla	1	11	-10605.80		
Mozilla	2	7	-10768.55	325.51	$< .0001$
D	1	11	-18009.84		
D	2	7	-18045.10	70.51	$< .0001$
K	1	11	-755.9808		
K	2	7	-809.3119	106.66	$< .0001$

The resulting tests are shown in Table 6. Columns show project, model (1 - the full model in Equation 3 and 2 - the simplified model from Equation 4), degrees of freedom, log-Likelihood, likelihood ratio, and p-value for the test comparing the full and the simplified models. For all the projects there is a significant difference between the simpler model in Equation 4 and the model in Equation 3, showing that fitting curves separately for different initial conditions significantly improves the model fit. In other words, the learning trajectories significantly differ among the three initial conditions, thus supporting Hypothesis 5.

7. LIMITATIONS

We have chosen familiar projects we studied earlier to ensure a good understanding of project's process and data collected from problem tracking systems. Therefore, this is not a random sample of all projects and that restricts the generalizability of our findings, but we have more confidence that these findings are not reflecting data artifacts unrelated to software development phenomena. The most effective way to generalize the results we obtained from the six specific projects is to reproduce them in multiple unrelated projects. However, the inaccessibility of the relevant data for most traditional projects may be an obstacle that is to some extent alleviated by the availability of open source project data. In this study we focused on using only the types of data that can be easily obtained from an issue tracking system of any project. That makes it simpler to reproduce our results in other projects. We also provide the open source project data, the minimally anonymized traditional project data and the scripts we used to anyone interested in replicating this study.

Human factors are difficult to measure because of the large variations among humans and the lack of understanding of how humans behave. This study tried to measure the social learning of programmers through the interactions they have in the MR workflows. One may argue about the extent to which it reflects the social behavior of

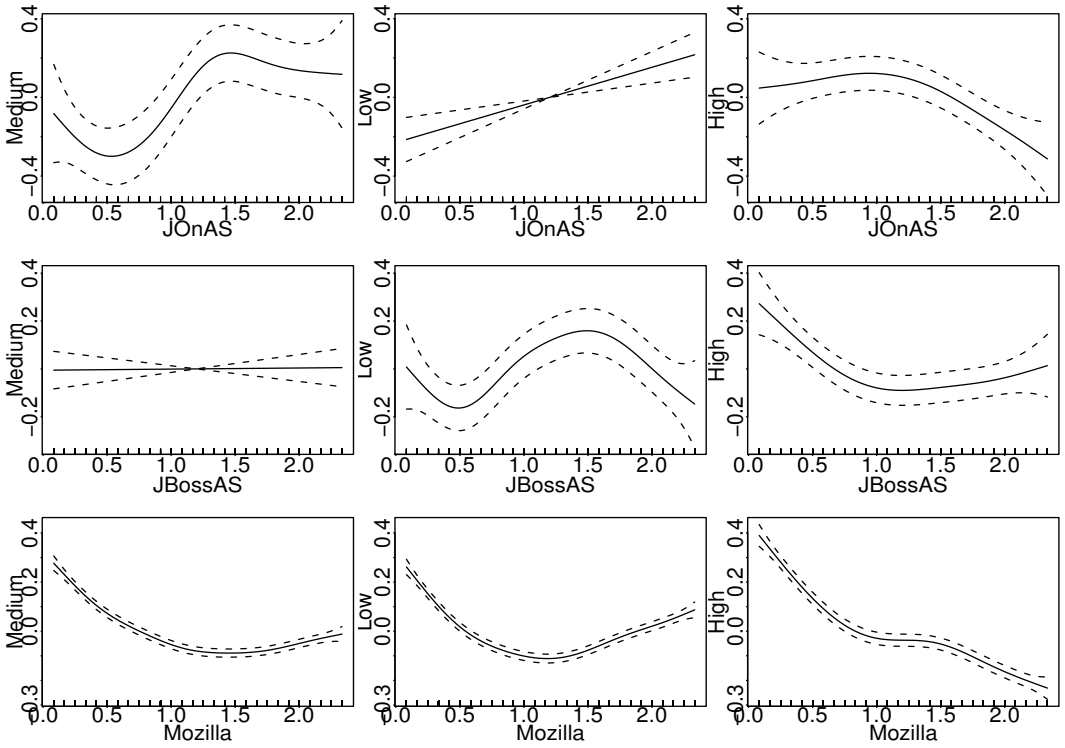

Figure 5: Estimates of RS trajectories. Conditions from left: medium, low, high. Projects from top: JOnAS, JBossAS, Mozilla

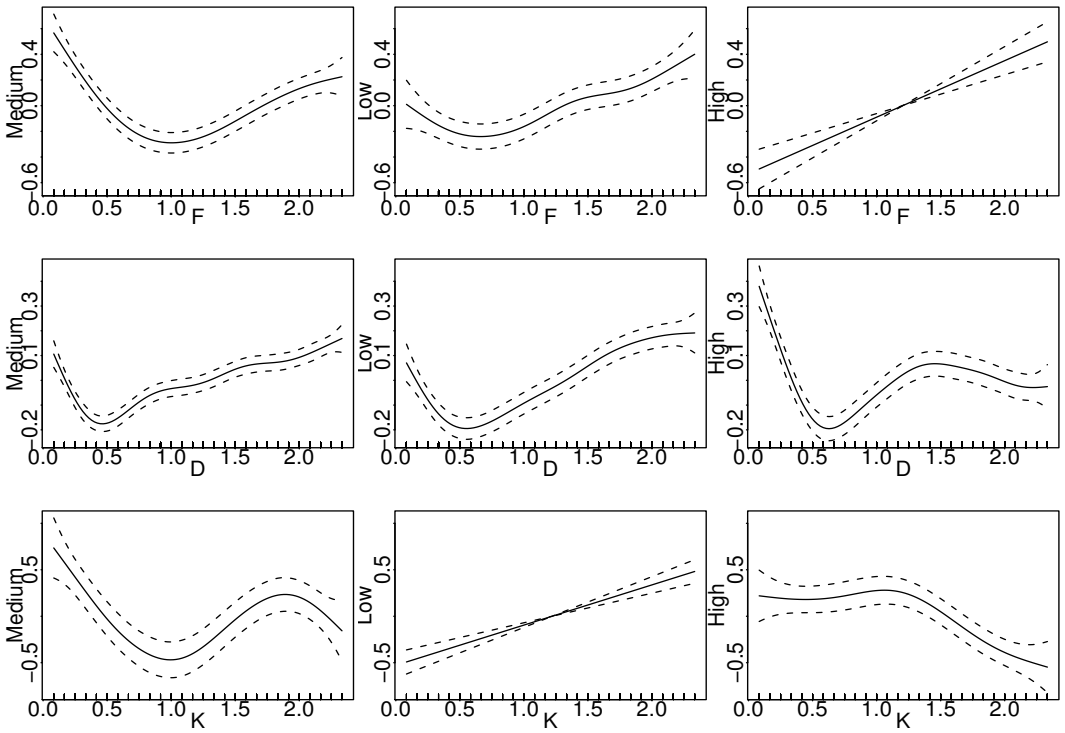

Figure 6: Estimates of RS trajectories. Conditions from left: medium, low, high. Projects from top: F, D, K.

programmers, but it is directly related to tasks and can be measured in most projects making them practical and replicable. Moreover, we operationalize measures in several ways and cross-check if the results are replicable. If results using multiple measures are similar, we have more confidence that the discovered relationship is robust with respect to these operationalizations. For example, to measure the project's RS using the individual developers' RS, we considered three measures, the first one is the ratio of the number of people all developers encountered in their MR workflows to the number of MRs all developers completed (in that month), the second is the arithmetic average of RS over all the developers, and the third is the geometric average (average the logarithms of RS) used in this study.

Data collection and cleaning was an effort intensive task, as outlined in Section 4, and various choices we made in cleaning the data may affect the results. In particular, we had several significant steps that required making a choice: a) identifying a set of contributor IDs corresponding to the same person — a filter map that transforms data by making all IDs used by the same person identical, b) identifying administrative IDs (IDs used by multiple people) (filter $rmadm$ that removes administrative IDs from data), and c) deciding when to measure the initial project environment for a developer (filter ts that determines the interval over which the project's RS is calculated). To make sure that these choices do not affect our results we have conducted multiple analyses. In particular, if we refer to the untransformed data as D, then we conducted analysis on $ts(D,k)$ and also on $map(ts(D,k))$, on $rmadm(ts(D,k))$, and on $map(rmadm(ts(D,k)))$ for $k = 6, 3, 0$ representing a six month project interval prior to joining, a six month interval starting three months prior to joining, or a six month interval starting at the time of joining. To simplify reproducibility we report the results obtained without ID mapping (map) and filtering ($rmadm$) for the RS calculated over the interval starting three months prior to joining and ending three months after joining $ts(D,3)$ (because we assume that the decision to join is made before making the first contribution and might be affected by the project's RS). There were only two differences from the reported data in the modeling results we observed over all these analyses. First, in the model for the intensity of joining contributors (Equation 2), the high-RS periods had significantly higher intensity compared to medium-RS periods as shown in Table 7 for the analysis on $map(rmadm(ts(D,3)))$. But

Table 7: Newcomer intensity (filtered data).

	Estimate	Std. Error	Pr(>|t|)
(Intercept)	5.52	1.35	0.00
Low	-1.82	0.38	0.00
High	0.57	0.18	0.01
D	2.05	1.43	0.18
F	0.63	1.66	0.71
JBossAS	2.42	1.40	0.11
JOnAS	0.15	1.83	0.94
Mozilla	5.50	1.35	0.00

the same coefficient for the unfiltered data shown in Table 4 does not have a significantly higher value as it would be expected according to Hypothesis 3. Second, with the cleaned and filtered data ($map(rmadm(ts(D,3)))$) Table 8 shows a significantly lower probability that developers joining during high-RS periods would become LTCs (Equation 1), unlike the coefficient fitted for the unfiltered data shown in Table 3. That strengthens Hypothesis 2. This analysis suggests that some phenomena may be strong enough to be visible even with severely degraded data (for example, there were 1.6K IDs in Mozilla that belonged to people who also used another ID at a later time). It is also worth noting that we excluded the period of the last 28 months from the study of LTCs, because it is impossible to determine who will become an LTC for anyone joining more recently.

Table 8: Probability of becoming an LTC (filtered data).

	Estimate	Std. Error	Pr(>|z|)
(Intercept)	-4.33	0.05	0.00
Low	1.47	0.08	0.00
High	-0.67	0.07	0.00
Trad.	1.83	0.22	0.00
D	1.57	0.22	0.00
F	1.36	0.24	0.00
JBossAS	0.28	0.15	0.06
JOnAS	1.44	0.27	0.00

8. CONCLUSIONS

We have tried to measure how the evolution of developers' socio-technical balance is impacted by the project environment they encounter while joining, with the hope of finding the hints of socio-technical congruence laws, and the hope of improving software production. We classified the initial project environment into three Relative Sociality types: high, low, and medium, and investigated how it affects the future trajectories of developers joining the project. We found that in both OSS and traditional projects, if the project is in a more social condition, more people per unit time tend to join; when the project has a low social environment, the newcomers are more likely to become long-term contributors. But only in OSS projects the intensity of people who would become LTCs is highest when the project is in its low sociality periods.

Our observations highlight the significant differences among the RS learning trajectories of the developers who join in low, medium, or high sociality environments and between OSS and traditional projects. This finding might serve as a hint for the developers to decide when to join a project depending on whom they want to become. It may also be helpful for the projects to design a training plan for newcomers to get people with different expertise needed by the project. For example, the trainers need to consider how to compensate people who join at "bad" times if they want them to make more meaningful contributions to the project and organization. Furthermore, the lessons from open source projects appear relevant to traditional projects. For example, when the project sociality measure is low, more newcomers per unit time would become LTCs. That may happen because they have more opportunities to get more deeply involved, have more appreciation of the project, for example, by being able to add major functionality or to be mentored by existing experienced developers, or simply are more likely to have a more genuine interest in the project.

It's interesting to observe how people become more productive: is it because of her nature driving her to achieve, the enforcing environment cultivating her (like the initial social climate investigated in this study), or a great master stimulating her? Software development is a knowledge-intensive task, it's more about how humans create things, and less about engineering. We might have to take a fresh look at the organizational science and cognitive science, and that might offer us the best opportunity to study and gain control over the largest source of influence on project performance, as Curtis [7] argued thirty years ago.

9. ACKNOWLEDGMENTS

We thank the National Basic Research Program of China under Grant No. 2009CB320703, the Science Fund for Creative Research Groups of China under Grant No. 60821003 and the Nature Science Foundation of China under Grant No. 61073016.

10. REFERENCES

[1] A. Begel, K. Y. Phang, and T. Zimmermann. Codebook: Discovering and exploiting relationships in software repositories. In *ICSE '10: Proceedings of the 32th international conference on Software engineering*, pages 125–134, New York, NY, USA, 2010. ACM.

[2] 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 '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 2–11, New York, NY, USA, 2008. ACM.

[3] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Trans. Software Eng.*, 35(6):864–878, 2009.

[4] M. Cataldo, P. Wagstrom, J. Herbsleb, and K. Carley. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *Conference on Computer Supported Cooperative Work CSCW'06*, Banff, Alberta, Canada, 2006.

[5] M. T. H. Chi. Active-constructive-interactive: A conceptual framework for differentiating learning activities. *Cognitive Science*, pages 73–105, 2009.

[6] M. E. Conway. How do committees invent? *Datamation*, 14(4):28–31, April 1968.

[7] B. Curtis. Substantiating programmer variability. In *Proceedings of the IEEE 69*, July 1981.

[8] J. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *2003 International Conference on Foundations of Software Engineering*, Helsinki, Finland, October 2003. ACM Press.

[9] J. Lave and E. Wenger. *Situated Learning. Legitimate Peripheral Participation*. Cambridge University Press, Cambridge, 1991.

[10] A. Mockus. Software support tools and experimental work. In V. Basili and et al, editors, *Empirical Software Engineering Issues: Critical Assessments and Future Directions*, volume LNCS 4336, pages 91–99. Springer, 2007.

[11] A. Mockus. Organizational volatility and its effects on software defects. In *ACM SIGSOFT / FSE*, pages 117–126, Santa Fe, New Mexico, November 7–11 2010.

[12] A. Mockus, R. T. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1–38, July 2002.

[13] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[14] K. Nakakoji, Y. Ye, and Y. Yamamoto. Comparison of coordination communication and expertise communication in software development: Motives, characteristics, and needs. In *Proceedings of JSAI-isAI2009 Workshop on KCSD2009*, pages 112–122. Springer Verlag, 2009.

[15] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[16] D. Redmiles, A. V. D. Hoek, B. Al-ani, T. Hildenbrand, S. Quirk, A. Sarma, R. Silveira, S. Filho, C. D. Souza, and E. Trainer. Continuous coordination: A new paradigm to support globally distributed software development projects, 2007.

[17] K. Schmidt and C. Simone. Coordination mechanisms: Towards a conceptual foundation of cscw systems design. *The Journal of Collaborative Computing*, 5:155–200, 1996.

[18] S. Wood. Low rank scale invariant tensor product smooths for generalized additive mixed models. *Biometrics*, 62(4):1025–1036, 2006.

[19] Y. Ye, Y. Yamamoto, and K. Nakakoji. A socio-technical framework for supporting programmers. In *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 351–360, New York, NY, USA, 2007. ACM.

[20] M. Zhou and A. Mockus. Developer fluency: Achieving true mastery in software projects. In *ACM SIGSOFT / FSE*, pages 137–146, Santa Fe, New Mexico, November 7–11 2010.

[21] M. Zhou, A. Mockus, and D. Weiss. Learning in offshored and legacy software projects: How product structure shapes organization. In *ICSE Workshop on Socio-Technical Congruence*, Vancouver, Canada, May 19 2009.

[22] S. Zyglidopoulos. Initial environmental conditions and technological change. *Journal of Management Studies*, 36:241–262, 1999.

11. APPENDIX

To facilitate replication of the results we provide a formal description of measures used in the study. Let developers in a project be represented by $i = 1, \ldots, N$, their time of joining as s_i, the time of their last MR-associated activity as e_i, and the total number of MRs i participated in as n_i. Denote the task (MR) by m and let j_m index the chronological order of MR status changes. Let $d(m, j_m)$ denote the developer involved in the action j_m and $t(m, j_m)$ be the time of action. Thus, the basic input for our analysis is a set of tuples $(m, j_m, d(m, j_m), t(m, j_m))$. Data filtering removes all tuples where $d(m, j_m)$ is an administrative login. Data mapping replaces $d(m, j_m)$ for developers with multiple logins by a single (canonical) login.

A workflow edge $W(i_0, i_1, t(m, j_m))$ is created for every pair from $i_0 = d(m, j_m)$ to $i_1 = d(m, j_m - 1)$ for all $m, j_m > 0$, and $d(m, j_m - 1) \neq d(m, j_m)$. Developer i's RS for a time period $[t_0, t_1]$ is

$$RS(i, t_0, t_1) = \frac{|\{i_w : W(i, i_w, t), t \in [t_o, t_1]\}|}{|\{m : d(m, j_m) = i, t(m, j_m) \in [t_o, t_1]\}|}$$

Project's RS for a time period $[t_0, t_1]$ is

$$RS(t_0, t_1) = \frac{\sum_i \log_i RS(i, t_0, t_1)}{|\{i : d(m, j_m) = i, t(m, j_m) \in [t_o, t_1]\}|}$$

Let the set of developers who participate for at least 28 months be denoted as $LT = \{i : e_i - s_i > 28\}$ and the 10-th percentile of their productivity $\frac{n_i}{e_i - s_i}$ as p_{min}. The set of LTCs is defined as $\{i : i \in LT, \frac{n_i}{e_i - s_i} > p_{min}\}$.