The Role of Mentoring and Project Characteristics for Onboarding in Open Source Software Projects

Fabian Fagerholm[1], Alejandro S. Guinea[1], Jürgen Münch[1], Jay Borenstein[2,3]
{fabian.fagerholm, azsanche, juergen.muench}@cs.helsinki.fi, borenstein@cs.stanford.edu

[1]Department of Computer Science
University of Helsinki
P.O. Box 68, FI-00014
University of Helsinki
Finland

[2]Department of Computer Science
Stanford University, Stanford
353 Serra Mall, CA 94305
USA

[3]Facebook
1601 Willow Road
Menlo Park, CA 94025
USA

ABSTRACT

Context: Onboarding is a process that helps newcomers become integrated members of their organisation. Successful onboarding programs can result in increased performance in conventional organisations, but there is little guidance on how to onboard new developers in Open Source Software (OSS) projects. **Goal:** In this study, we examine how mentoring and project characteristics influence the effectiveness and efficiency of the onboarding process. We study a collaboration program involving a total of nine Open Source Software projects and more than 120 students from different universities around the world as part of Facebook's Education Modernization Program. **Method:** We use quantitative measurements of source code repositories, issue tracking systems, and discussion fora to examine how newcomers become contributing members of their OSS projects. **Results:** We found that developers receiving deliberate onboarding support through mentoring were more active at an earlier stage than developers entering projects through conventional means. Also, we found that project size and lifetime influenced onboarding. **Conclusion:** Empirical decision support can contribute to a more effective onboarding process in OSS projects. Mentor support in critical stages can accelerate the process, but project maturity is also a significant factor that increases the effect of onboarding.

Categories and Subject Descriptors

K.6.1 [**Project and People Management**]: [Staffing; Training; Systems development]

General Terms

Experimentation, Human factors, Management

Keywords

Onboarding, Organisational Socialisation, Open Source Software, Case Study, Mentoring, Software Teams, Distributed Development

1. INTRODUCTION

Onboarding, also known as organisational socialisation, refers to processes that help new employees learn the knowledge, skills, and behaviours they need to succeed in their new organisations [5]. Newcomer adjustment has been associated with important outcomes such as satisfaction, commitment, turnover, and performance [4]. In environments with high employee turnover and high cost of replacing an employee, well-functioning onboarding procedures can give organisations a competitive advantage: by correctly applying onboarding programs, software development groups can be grown and their productivity increased without needlessly holding back existing members or disrupting normal activities [19].

While onboarding is well understood in the context of conventional organisations [5], it is not obvious how to transfer this understanding to new forms of organisation. Software companies increasingly turn to Open Source Software (OSS) projects as a source of low-cost innovation and productivity [10, 28, 42]. OSS projects are often at the core of ecosystems which companies must become involved in to acquire a market share. The scale of projects may exceed the capabilities of a single company, and customers may only be reachable through a certain OSS ecosystem. This creates a need for onboarding both the company's own employees into existing OSS projects, as well as external individuals into the company's own OSS projects.

Despite these potential benefits, guidance on how to manage OSS projects is still largely missing in many areas [15]. To gain the benefits of OSS, organisations must understand the particular traits of OSS development that set it apart from other kinds of approaches. In particular, the nature of the development process followed in many OSS projects poses a number of challenges that can determine to what extent benefits are gained. One important aspect of managing OSS projects concerns developer involvement, engagement, and productivity. Systematic onboarding is not a common practice in OSS development. The inclusion of newcomers is often seen as depending mainly on self-motivation and personal interest [37] rather than as a systematic process for quick integration. This is an important consideration for organisations wishing to support new OSS developers.

In this paper, we present an empirical multiple-case study in which we investigate onboarding in OSS projects. Our goal is to analyse factors that have an influence on the efficiency and effectiveness of onboarding in OSS projects. We examine and compare OSS projects and OSS developers in the context of a collaboration program conceived and organised by Facebook and Stanford University. In this program, students from different universities around the world were selected and assigned to one of nine different OSS projects. The program provided several elements of deliberate onboarding: throughout the collaboration program, a mentor from each project helped with integrating the students, and there was a common, co-located kick-off Hackathon event at the beginning of the collaboration. We specifically investigate how developers receiving onboarding support as part of the program perform compared to other developers, and how the performance of supported developers in the different projects compare to each other. We obtain empirical data for assessing the efficiency and effectiveness of onboarding in OSS projects.

Our analysis revealed that as a whole, developers receiving onboarding support were more efficient and effective than developers who entered the same projects through the natural, non-deliberate process which is common in many OSS projects. We also found that there were differences in onboarding results which could be attributed to OSS project characteristics, such as project age and distribution of contributions among project members.

The contributions of this work include (1) the empirical observation that deliberate onboarding practices have an impact on the integration of new developers in OSS, and (2) insights into how mentoring and project characteristics can influence the speed of integration. In addition, we offer some suggestions regarding onboarding for OSS communities and companies interested in getting involved in OSS projects.

The remainder of this paper is structured as follows. In the next section, we present related work on onboarding and the research questions. Section 3 provides a detailed description of the collaboration program that forms the context of our study. The study design, along with the definition of metrics to assess onboarding efficiency and effectiveness, is presented in Section 4. In Section 5, we present and analyse the results. The implications of the results are discussed in Section 6. Limitations and threats to validity are discussed in Section 7. Finally, we summarise the findings and outline future work in Section 8.

2. RELATED WORK AND RESEARCH QUESTIONS

Research into onboarding processes in the context of OSS projects is currently scarce. Most related studies deal only with the natural, non-deliberate process that developers typically follow when joining an OSS community (e.g. [8, 32, 23, 37]). They examine a process where joining and participating in OSS projects is a personal initiative driven by individual motivation and interests. This natural process of involvement in OSS stands in contrast to a deliberate onboarding process that is driven by organisational goals and that utilises guidance and support mechanisms provided by the organisation. To our knowledge, only a previous paper by the present authors – with preliminary results from this study – treats the process of onboarding in OSS directly [21]. In that preliminary analysis, we focus only on the influence of mentoring and consider only the first 12 weeks of the development activities. However, several other works present theory and raise relevant considerations for studying onboarding.

Ducheneaut [18] approaches onboarding from a sociological point of view, considering the perspective of individual developers. This study found that joining an OSS project requires newcomers to go through a complex socialisation process. Only those who have managed to define and present themselves as skilled and well versed in development will reach the official status of developer in a project. Hence, contributing to an OSS project is as much a process of socialisation as a demonstration of technical expertise. Von Krogh et al. [44] gather different aspects related to the process of joining an OSS project, providing observations and suggestions that can serve to build controlled joining mechanisms. They found that the participation of developers in OSS projects is not costless, and detailed some of the community-related benefits to newcomers. They pointed out the importance of devising a theory including these and other elements that have not been apparent in the study of OSS development.

More general perspectives on onboarding in software companies have been presented in the literature. Typical project characteristics and context factors, and their impact on onboarding, have been studied [17], revealing that human guidance is more important than any other factor for helping newcomers to settle in and providing an understanding of source code and project issues. Onboarding newcomers into virtual teams has been analysed [29], as well as novice software developers and new college graduate students heading for their first job [6, 7]. These studies have found that poor communication and lack of experience in socialisation skills are at the root of most problems newcomers encounter in their development endeavours.

Mentoring has been identified as a key activity to support onboarding. Swap et al. describe mentoring as a fundamental knowledge transfer mechanism in companies [40]. Sim and Holt present a study of mentoring patterns that occur in the process of integrating new developers into software projects [19], finding that mentoring is an effective way to onboard newcomers and to overcome issues derived from lack of good system documentation. However, it can also result in inefficiency due to decreased productivity of developers performing the mentoring function. In the context of OSS, some studies have pointed out the importance of mentoring for sharing and reuse of knowledge [39, 15, 38, 43]. More recently, Shilling et al. directly studied the impact of mentoring on training and retention of developers in OSS projects [36]. Based on their findings, they propose mentoring as a training method for OSS projects, and devise a measuring scheme for evaluating the appropriateness of mentoring for facilitating learning and retention among developers.

Inspired by the theory and findings presented above, our first research question relates to the aspect of human guidance in the form of mentoring:

RQ1 What is the influence of mentoring on the efficiency and effectiveness of onboarding in OSS projects?

Besides actively supporting the onboarding process with measures such as mentoring, companies can often choose between different OSS projects they could turn to. A criterion for choosing an appropriate OSS project could be the ease of getting integrated in the project and getting a sufficient number of developers effectively and efficiently onboard. Therefore, we also consider selected characteristics of OSS projects that might have an impact on onboarding of new developers. Significant attention has been given in previous studies to the type of license projects are using [15]. However, based on the observations of a large number of OSS projects made by Capiluppi et al. [12], it appears more appropriate for our purposes to analyse other project characteristics, such as size, age, and the distribution of developer contributions. Our second research question relates to the relationship between project characteristics and onboarding:

RQ2 What is the influence of specific OSS project characteristics on the efficiency and effectiveness of onboarding in those projects?

Different metrics have been proposed to measure the performance of OSS developers both over time and with respect to their overall project contribution. Several previous studies have devised metrics based on commits – individual items of contribution in source code repository data [2, 27]. Zhou and Mockus have devised a metric for assessing whether developers have achieved a high level of proficiency ("true mastery") [46]. Fritz et al. have proposed a metric for measuring the degree of knowledge, based on a combination of authorship and interest in particular parts of software projects, where the interest is measured by the amount of selections and edits over a source code element [25].

An additional consideration in OSS research is sampling. Some previous OSS-related studies have focused on a small number of projects selected based on their large user base, widespread use of their software, or due to interest in some specific aspects of the project (e.g., [33, 11, 43, 2]). Focusing on one or two projects can yield deep insights into their particular characteristics, but might limit the generality of the findings. Other studies have included a larger number of projects with heterogeneous characteristics, providing a higher level of generality and validity. One good example of the latter is Capiluppi et al. [12], which studies over 400 OSS projects.

The availability and collection of OSS project data poses specific challenges to researchers. Several previous studies have examined the kinds of data available through publicly available source code repositories, methods for extracting and cleaning it, and approaches to analysing it (e.g., [16, 33]). While using this data is tempting due to ease of access, there are several caveats involved. Research into mining software repositories warns against blindly using data from large amounts of projects hosted on source code hosting sites and from distributed version control systems (e.g. [30, 9]). The data is often incomplete both for technical reasons and because important project events may occur outside the publicly available data set, e.g., in personal conversations both online and in person.

3. CONTEXT

During the first half of 2013, Facebook, Inc. implemented an OSS collaboration as part of their Education Modernization Program. The program was initially piloted by Jay Borenstein, Education Modernizer at Facebook and lecturer in Computer Science at Stanford University. Based on the success of the initial pilot, the program was expanded to introduce the element of cross-university collaboration. This program forms the context for our study. The program brought several OSS projects, companies, and universities around the world together to provide students with the opportunity to experience OSS development and learn relevant related skills.

More than a dozen universities from different countries were invited to participate in the collaboration program. Each of them integrated the program into its curriculum according to the local academic calendar and technical infrastructure. For example, at the University of Helsinki, the program was integrated as an advanced master's-level course conducted in the Software Factory laboratory for empirical software research and education [1, 22]. Students could choose to participate for a duration of three to six months, and with a varying degree of working hours per week.

A total of nine OSS projects were selected into the program by Mr. Borenstein. The main criterion for inclusion was the availability of a mentor who was familiar with the project and who could devote time to supporting the students. The participating OSS projects were Freeseer, Kotlin KJ2K, MongoDB, Mozilla Open Badges, Phabricator, PouchDB, Review Board, Ruby on Rails, and Socket.IO. From these, our study focuses on Kotlin KJ2K, Phabricator, Ruby on Rails, and Socket.IO. We decided to focus on these projects because the student teams included members from the Department of Computer Science, University of Helsinki, and we were thus able to observe them more closely.

Approximately 120 students participated in the collaboration program. After an admission process that varied between universities, these students indicated their preferred OSS projects, and were then assigned to project-specific teams with members from at least two universities. Each OSS project was assigned two teams, and each team had four to eight students. An experienced developer from each project functioned as a mentor for the student teams.

Mentors were expected to be responsive to students and had demonstrated proficiency as software engineers, with a large number of contributions to the OSS projects they were associated with. No previous experience as mentors was required from them, and no written agreement or contract was made between them and any other party. The program followed the typical configuration of OSS development, where people participate, maintain personal engagement, and share their knowledge more for intrinsic or altruistic reasons than for other kinds of incentives [39, 34, 23].

At the beginning of the collaboration program, a three-day kickoff Hackathon was organised at Facebook's offices in California, USA. During this event, teams were formed and each student got to know their team mates and their OSS mentors. The mentors provided the students with practical training required to start contributing to their OSS projects. This included familiarisation with the code base, tools, and procedures used in the project.

After the kickoff, students returned to their home universities to continue working for their assigned projects, now as teams distributed over different geographical locations and time zones. Some practices were suggested in order to support successful onboarding of the students. Among these were daily meetings and communication via email, chat, and video conference, as well as the establishment of high-level goals after a period of familiarisation.

The mentors performed different activities for supporting students, such as participating in online fora and mailing lists with the students, conducting, or sometimes only participating in online meetings through video conferencing and chat, helping students find and understand tasks, reviewing code contributions and providing feedback on them, and helping to coordinate the work of students through issue tracking systems.

Students were in general free to work on any tasks they found relevant. Initially, mentors led students to work on small tasks suitable for novices, to make them become proficient little by little, until they had the skills and knowledge necessary to tackle more complex, self-selected tasks. The tasks were programming tasks of different types and complexities, from bug fixes to more advanced activities involving design and implementation of new features.

Overall, students were onboarded into the OSS projects and communities in ways that were common for each project. Hence, they were exposed to the regular norms and implicit policies of the corresponding community. All these factors provided the students with a realistic scenario where they were able to work as regular developers.

4. STUDY DEFINITION AND DESIGN

Our *goal* is to analyse factors that have an influence on the efficiency and effectiveness of onboarding in OSS projects. Specifically, we examine mentoring (RQ1) and selected characteristics of OSS projects that might be relevant for the process of onboarding (RQ2). We aim to establish a relationship between these attributes by conducting quantitative data collection and analysis. Since we are conducting an observational study that examines several OSS projects in their natural context, this work can be characterised as an empirical multiple-case study, as defined by Wohlin et al. [45].

We use the Goal-Question-Metric (GQM) approach [3] to operationalise the research questions into two corresponding measurement goals:

G1 Characterise and understand the effect of mentoring on the efficiency and effectiveness of the onboarding process from the point of view of project managers in the context of the OSS collaboration program.

G2 Characterise and understand the effect of project characteristics on the efficiency and effectiveness of the onboarding process from the point of view of project managers in the context of the OSS collaboration program.

By project managers, we mean individuals who are responsible for managing the introduction of new members into OSS projects. In the remainder of this section, we discuss the conceptualisation of mentoring, project characteristics, and efficiency and effectiveness of onboarding, as well as the derivation of questions and metrics associated with the measurement goals shown above.

The projects under study manage their development and communications activities through the GitHub system, which provides publicly accessible tools for collaborative software development: a version control system for program source code, an issue tracking system, and discussion tools for coordination. The data set of this study is available for download from the Figshare repository [20]. It can be used, for instance, for replications and further analyses.

4.1 Conceptualisation of Factors in Onboarding

Mentoring. The role of the OSS mentors was to support the developers in most of their activities – e.g., recommending and helping to interpret tasks, explaining the architecture of the software, and assisting in technical development details. Because of the influence the mentors had on developers, we assume that positive results in developer performance is related at least to some extent to the support given by the mentors both during the kickoff Hackathon and over the duration of the whole collaboration program. The effect of successful mentoring is likely to be visible as an increase of specific developer actions. For instance, a mentor suggesting suitable tasks to perform could increase the number of commits over time by helping developers to focus on specific activities and become proficient at an increased pace. Also, mentors can encourage developers to cope with certain development and communication activities, thus influencing the degree of collaborative activities between developers. Higher activity among developers reflects to some extent the influence that mentors have on developer performance. This leads to our first measurement question:

Q1 To what extent does mentoring influence the progression of active participation of developers in OSS projects? (G1)

The distribution of contributions in OSS projects usually follows a power-law distribution, with a few developers contributing most, and the majority contributing only a little (e.g. [13, 33]). Newcomers typically start at the periphery of the project community, and may have little contact with core developers [14]. We hypothesise that expert support from core project members can help newcomers by influencing their motivation, increase cohesion among community members, and reduce the gap between required skills and knowledge and developer ability. As mentioned in Section 3, the mentors were well versed in their corresponding projects and willing to help developers on a regular basis. They acted as experts who might push developers into higher levels of motivation, engagement, proficiency, and consequently higher levels of contribution. If developers receiving mentoring become more active contributors than the peripheral developers [31] of the corresponding project, this can be seen as stemming from the positive influence of mentoring.

We analyse the influence of mentoring on levels of contribution by comparing the performance of developers that have been deliberately onboarded as part of the collaboration program against the performance of developers that have joined the same OSS projects by the natural process of involvement. This comparison is formulated as the following measurement question:

Q2 To what extent does mentoring influence the overall contribution of developers in OSS projects? (G1)

Project characteristics. Even if deliberate onboarding support is given, its influence on the efficiency and effectiveness of the onboarding process may vary between projects. OSS project characteristics may moderate the effect of interventions such as mentoring. The OSS project characteristics that are analysed in this study were chosen based on their relation to and potential impact on onboarding. Project characteristics may influence the dynamic aspects of the onboarding process, leading to our third measurement question:

Q3 To what extent do specific project characteristics influence the progression of active participation of developers in OSS projects? (G2)

On the other hand, project characteristics may have an effect on the total outcome of the onboarding process at some specific point in time:

Q4 To what extent do specific project characteristics influence the overall contribution of developers in OSS projects? (G2)

We consider characteristics that account for project size in terms of contributions, number of developers, and relative importance of their corresponding peripheral developers. In addition, we consider the age of the project and the level of interest that the project has attracted from developers both within and outside the project. Characteristics related to size can help establish a baseline, or normal level of developer proficiency. The maturity of the processes and procedures of the project – regardless of whether they are documented or not – might be related to its lifetime. This, in turn, may have an impact on how quickly and easily new members can become integrated. The appeal of a project might be relevant to measure the potential growth of the contributor population. Also, and probably more importantly for onboarding, an appealing project may be more likely to maintain developer motivation.

Next, we describe each selected characteristic, justify the choice, and explain its purpose and use in our study.

Commits. This refers to the total number of commits that have occurred within a project during a specified time interval and for a specific set of developers. This characteristic can reflect both the size of the project and the degree of active participation in the form of code-writing by contributors. For the onboarding process, this characteristic can be relevant in evaluating the performance of newcomers compared to the peripheral developers.

Contributors. This characteristic is concerned with the total number of contributors in a project. It is another size-related characteristic that reflects organisational or group size rather than, as in the case of commits, the size of a stream of activity.

Appeal. We conceptualise the appeal of a project as the observable interest that current and potential developers have shown towards it. Specifically, appeal is obtained by counting the number of persons

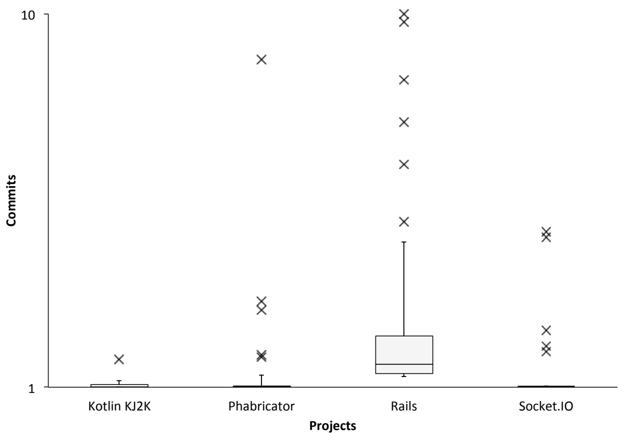

Figure 1: Distribution of commits.

who have used the GitHub function to "follow" a project, plus the number of times the project has been "forked", or copied to initiate a new development branch. This characteristic can help assess how much attention the project is receiving from the OSS community, beyond directly observable contributions and online discussions. It can be relevant in understanding the performance of the developers within a project, given that a great part of it depends on self-motivation and interest.

Lifetime. This characteristic refers to the time that the project has been active. This characteristic is aimed to reflect the maturity of a project, although for a full assessment of maturity, other aspects such as the history of the project in terms of contributors and contributions and detailed evaluation of project practices have to be considered as well. A mature project is expected to benefit a smoother inclusion of developers.

Peripheral contribution. This characteristic refers to the proportion of contributions made by the peripheral developers, in contrast to contributions made by all developers. We consider peripheral developers as those developers who have made some contribution to the project but who are not "core" developers. Being able to distinguish between peripheral and "core" developers allows us to assess the relative level of contributions from the developers receiving onboarding support. Thus, we can determine the level of contributions that developers with onboarding support should reach in order to consider the onboarding successful.

Efficiency and effectiveness of onboarding. We consider the efficiency of an onboarding process to correspond to the time taken to reach a specific level of integration. A more efficient process shortens the time it takes to learn the skills and obtain the knowledge required to contribute to organisational goals. By effectiveness of onboarding, we mean the depth of integration. A more effective process results in deeper integration and a greater likelihood of contributions overall as well as a greater likelihood of more valuable contributions.

To evaluate the influence of mentoring on onboarding efficiency and effectiveness, we compare the speed of contribution and overall contribution of developers who received deliberate onboarding support to a random sample of peripheral developers who did not participate in the collaborative program and thus can be assumed to have joined through the non-deliberate joining process that is natural for each OSS project. To evaluate the influence of project characteristics, we compare the speed of contribution and overall contribution of each group of developers who received deliberate onboarding support in each project to each other. For systematic comparison, we first identify the "core" developers of each project by analysing outliers in the distribution of contributions. Following a common convention for boxplots [24], we consider outliers to be points exceeding $Q3 + 1.5 \times IQR$. We do not consider them in the analysis, since successful onboarding does not require developers to be at the highest level of expertise, or, in our case, to become a "core" developer. Rather, they should be able to contribute to the project on a regular basis at the level of proficiency they have acquired. Once the core developers are removed from consideration, we obtain the distribution of commits of the remaining developers (i.e., the peripheral developers). This choice is supported by previous studies (e.g., [14, 13, 33]). By comparing developers with deliberate onboarding support to the peripheral developers, we adjust the expected level of contribution to be within normal ranges for regular active participation.

5. ANALYSIS AND RESULTS

In this section, we present the results that aim to answer the GQM questions formulated in Section 4, and consequently help to answer the research questions of our study. Before describing the results, we explain how we obtained the population corresponding to the peripheral developers of each project according to the criteria given in Section 4.1.

The criteria stated that the peripheral developers group is formed by the first three quartiles of the distribution of overall commits of developers, with the core developers removed from consideration. We analyse the distribution of contribution of the OSS projects considered in our study using a box-and-whisker plot, shown in Figure 1. The distribution of commits for each project, scaled to the range $[1, 10]$ and depicted using logarithmic scale, can be seen together with all the outliers. The box-and-whisker plot helps us to find core developers and recognise the shape of the commit distribution of each project.

Analysing Figure 1 in more detail, we can find useful information for comparing the relationship between project characteristics and the efficiency and effectiveness of onboarding. For each project's distribution we obtained the Gini coefficient in order to assess how evenly distributed they are. The Gini coefficient expresses the inequality of a distribution as a value between 0 (complete equality) to 1 (complete inequality) [26]. Kotlin KJ2K, with a Gini coefficient of 0.69, and Ruby on Rails, with 0.70, are more evenly distributed. In contrast, Socket.IO, with a Gini coefficient of 0.90, and Phabricator, with 0.91, display a very high inequality among the contributions of their developers. The behaviour of the Gini coefficient can be seen in the figure.

All comparisons that follow from here are made either with respect to the peripheral developers group, characterised by the analysis of the commit distribution, or based on the shape of the distribution shown by each project.

5.1 Influence of Mentoring on Onboarding

RQ1 What is the influence of mentoring on the efficiency and effectiveness of onboarding in OSS projects?

We now discuss the results that answer the GQM questions corresponding to the first research question:

Q1 To what extent does mentoring influence the progression of active participation of developers in OSS project?

To answer this question, we compare the results of two selected groups during a predefined period of time. The time window for

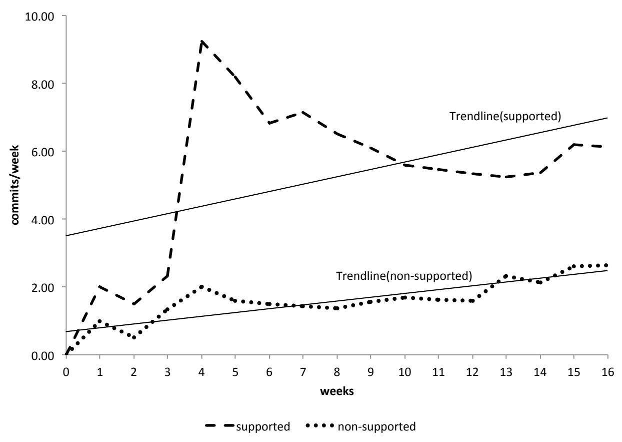

Figure 2: Speed of contribution.

the comparison between developers with onboarding support and peripheral developers without support corresponded to the duration of the collaborative program, i.e., 16 weeks. In order to obtain the results corresponding to developers with onboarding support, we register them from the beginning of the collaboration program until its end. For the non-supported developers, we identified the time stamp of their first commit, and then counted metric values beginning one week before this date and for 16 weeks thereafter. The reason for this choice is that based on observation of the developers with onboarding support, the first commit may not be registered immediately when the developer begins to participate in a project. At the same time, using the time stamp when the developer has signed up to participate in the project may not adequately reflect when the developer has actually begun active participation, with a large gap between sign-up and first activity.

The results of the speed of contribution over time exhibited by developers with onboarding support compared to non-supported developers are shown in Figure 2. An analysis of the progression of the two groups shows a noticeable positive trend in both. The speed of contribution among supported developers is visibly higher from the beginning. We used a two-tailed paired samples t-test to check for the statistical significance of the difference between the speed of contribution of each group (i.e., H_0: same speed), obtaining a t value of 7.38 – higher than the critical value of 2.12 – with a significance level of 5%. Thus, we can say that the difference between the speed of contribution of supported developers and non-supported developers is statistically significant.

The varying shape of the plots in in Figure 2 likely reflects the nature of human learning. When encountering a previously unknown task, we expect that developers engage in learning activities such as gathering and interpreting information to clarify the task and understand the software they are about to modify. Once they have accumulated enough knowledge and have reached a satisfactory level of understanding, they can begin to perform actual visible work. At some point, the performance of developers reaches a point of stabilisation where the speed converges around a constant value [46]. This stabilisation stage can be seen in the last weeks of the time window under analysis.

Based on the comparison between the speed of contribution of the two groups, it is clear that the supported developers surpass the performance of the non-supported developers. It seems reasonable to say that it has a positive influence on the performance of developers, since it is a main factor that separates developers receiving deliberate onboarding support from those not receiving it. Furthermore, since the speed of contributions for developers receiving mentoring is significantly higher, we can safely say that in the context of OSS development, mentoring is one element that influences the efficiency of onboarding.

Q2 To what extent does mentoring influence the overall contribution of developers in OSS projects?

The numbers concerning the overall contribution of supported developers and non-supported developers per project displays a clear advantage for the supported developers [20]. The supported developers contributed a total number of 11 commits in Kotlin KJ2K, 26 commits in Phabricator, 40 commits in Ruby on Rails, and 21 commits in Socket.IO. In contrast, the group of non-supported developers contributed a total of 5 commits in Kotlin KJ2K, 11 commits in Phabricator, 17 commits in Ruby on Rails, and 9 commits in Socket.IO. In all cases, the contribution of supported developers is about three times larger than the contribution from non-supported developers. We can thus consider the onboarding process as being successful. As with the previous GQM question, we can say that mentoring has been an important element in obtaining such good results. Our results show that mentoring has had a positive influence on the onboarding process in the case context. Therefore, mentoring may positively impact the effectiveness of an onboarding process in the context of OSS projects at large.

5.2 Influence of Project Characteristics on Onboarding

RQ2 What is the influence of relevant characteristics of projects on the efficiency and effectiveness of onboarding in OSS projects?

To answer this question, we first select a number of relevant characteristics according to the criteria in Section 4. To facilitate comparison, appeal was normalised to be in the range [0, 1]. Projects with lower appeal are given values closer to 0 and projects with higher appeal are given values closer to 1.

Table 1 shows the results obtained for project characteristics. Ruby on Rails has the highest numbers on all characteristics except peripheral contribution, where it comes second. The numbers indicate that Ruby on Rails could be a project that benefits the onboarding of developers. Given its lifetime, the maturity of the project seems to impact the number of commits, number of contributors, appeal, and, to a lesser extent, the proportion of the average contribution of the peripheral developers. The highest peripheral contribution, that of Kotlin KJ2K, can be explained in part by its low number of contributors, which makes the contribution of each developer more important. This can also be corroborated by the number of commits per developer of Kotlin KJ2K, shown in Figure 1 and supported by its Gini coefficient which, as previously shown, is more evenly distributed than Phabricator and Socket.IO.

The projects Phabricator and Socket.IO have similar characteristics. However, Phabricator is somewhat higher in terms of commits, contributors, and appeal, but still has almost the same level of peripheral contribution. This can be explained by a large outlier in the Phabricator project, described at the beginning of this section, which makes the contributions of the peripheral developers in Phabricator be proportionally smaller than the one of Socket.IO.

Based on the characteristics obtained for each project, we answer RQ2 by providing results and analysis corresponding to our GQM questions *Q3* and *Q4*, as defined in Section 4.

Q3 To what extent do specific project characteristics influence the progression of active participation of developers in OSS projects?

Table 1: OSS Project Characteristics

Project	Commits	Contributors	Appeal	Lifetime	Peripheral Contribution
Kotlin KJ2K	123	10	0.00	1.0	0.10
Phabricator	4876	164	0.11	2.4	0.05
Ruby on Rails	20000	2201	1.00	8.5	0.08
Socket.IO	1469	55	0.05	2.1	0.06

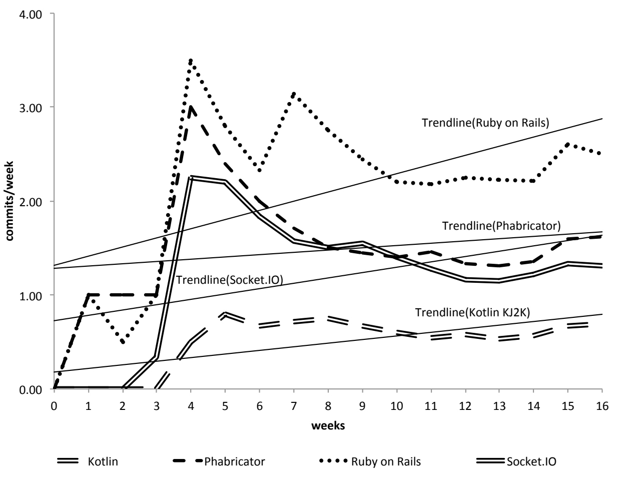

Figure 3: Speed of contribution per project.

Based on the 16-week time window established for answering *Q1*, Figure 3 shows the progression of the speed of contribution over time of the developers with onboarding support for each of the projects.

Ruby on Rails exhibits an overall higher speed than the other projects. The opposite case is Kotlin KJ2K with the lowest speed. Between these, Socket.IO falls slightly below Phabricator, which exhibits an initial speed comparable to the one shown by Ruby on Rails.

It is important to notice that the speed values shown in Figure 3 do not depend on differences in population size, since the number of developers that received onboarding support was roughly the same for all projects. Therefore, the results show that for developers receiving onboarding support, those participating in Ruby on Rails had higher speed values than those participating in any of the other projects. We can thus say that project characteristics have an influence on the speed of development and how fast a certain level of performance is reached.

Q4 To what extent do specific project characteristics influence the overall contribution of developers in OSS projects?

Looking at the overall number of commits from supported developers on each project (i.e., Kotlin KJ2K (11), Phabricator (26), Ruby on Rails (40), and Socket.IO (21)), and considering the characteristics of each project, some patterns become apparent. The appeal of a project, its size in terms of number of commits and number of contributors, as well as its maturity based on lifetime, appears to be related to the level of contribution of developers. Ruby on Rails has a considerably higher level of contribution compared to the other projects, and its characteristics are also higher than the other projects. Conversely, Kotlin KJ2K is at a considerably lower level on all traits than the other projects.

The proportion of peripheral contribution is an equaliser in the sense that it evens out the contribution distribution. This can be corroborated by comparing the results for Phabricator and Socket.IO. While the former has a slightly higher number of commits, the difference is very small; Phabricator is slightly higher on the commits, contributions, and appeal characteristics. In the case of Kotlin KJ2K, however, the effect is not large enough to compensate for the small project size.

Based on our analysis, the selected characteristics appear to influence both the efficiency and the effectiveness of onboarding in the OSS projects considered. In general, higher numbers in size (i.e., numbers of commits and number of contributors), appeal, and lifetime is related to an improvement in the performance of new developers. In addition, it has become apparent that the average contribution per developer of the peripheral developers of a project is a characteristic that can influence the onboarding of newcomers, since it may be an indicator that any developer, not only core ones, have greater opportunities to influence the overall performance of the project.

6. DISCUSSION

Mentoring influences the efficiency and effectiveness of onboarding in OSS projects. Having access to a "core" developer appears to be highly beneficial for new developers. Establishing a relationship with these developers could therefore be a strategy for onboarding company employees in important OSS projects. However, based on our observations during the collaborative program, mentoring requires a significant investment of time and effort. Practitioners in a project manager or similar role must balance the cost against the benefits both in terms of how much their own organisation can spend and expect to benefit, but also in terms of how to attract mentors from within the OSS project. As a compensation for mentoring efforts, some benefit should be offered to the mentor. One concrete example is offering to work on tasks that are otherwise neglected. Also, limiting the duration of the mentorship may make it more appealing. For OSS projects wishing to accelerate inclusion into their developer community, offering mentorship to prospective new developers might be a viable strategy. However, we recommend that mentorship should not bypass the usual requirements of developer proficiency and quality control in OSS projects. One example of a successfully operating mechanism for including new developers into OSS projects is the Debian project's New Member process [41]. That process includes many elements of a deliberate onboarding process although it does not focus on deliberate mechanisms for accelerating inclusion of new developers.

Mature projects are better equipped to receive new developers. In mature OSS projects, the community is concerned not only with the immediate goal of producing software, but also with the sustainability of the project itself. In such projects, some effort is devoted to addressing developer turnaround and needs for additional developers. Our results indicate that such projects are already better equipped to receive new developers even without deliberate onboarding support. This has two important implications: deliberate onboarding may have a greater effect in mature projects, and less mature projects may benefit from onboarding if it can be integrated into the normal culture of the project, thus increasing its maturity. For OSS projects, this means that participating in programs which assist with onboarding can be beneficial. For companies, it means that the maturity of the project should be taken into account when designing an onboarding program. Particularly, the goals of the onboarding program may have to be adjusted depending on the maturity of the project – either more towards improving the project maturity in less mature projects, or more towards specific inclusion targets in more mature projects.

Onboarding support may occur through communication channels that are not publicly visible. In our results, we found that the progression of speed of contribution proceeded in bursts (see Figure 2). While the exact cause is unknown, we hypothesise that this may be related to the nature of human learning, or the use of private communication channels. In light of this, we offer some observations regarding the nature of mentoring that may be important to consider. The mentoring relationship includes both elements of teaching knowledge and skills, but also elements of mutual reflection, especially regarding the norms, culture, and social aspects of OSS projects. Mentoring may require a private space where such considerations can be discussed without needless interference from the rest of the project. For project managers responsible for onboarding, this means that they must trust mentors to perform their work even though parts of it may be invisible. Successful onboarding will then be visible in concrete project results.

7. THREATS TO VALIDITY

Several factors threaten the validity of our findings. Following Runeson et al. [35], we discuss validity through four aspects: construct, internal, and external validity, as well as reliability.

Construct validity concerns the degree to which a measure actually measures what it claims to measure. Our method of assessing the efficiency and effectiveness of onboarding can represent a threat to the validity of our study, since it compares two processes which are different in nature. Onboarding processes can be considered as deliberate, well-structured processes that aim to get developers engaged and proficient in their projects in the shortest possible time. The non-deliberate, natural process of involvement in OSS projects can be thought of as relying primarily on the self-motivation and interest that prompt developers to participate. One necessary step in validating a specific onboarding process design should include a comparison of different onboarding processes in order to obtain a more precise evaluation of the factors that affect its behaviour and success. Many OSS developers participate in their projects due to personal reasons and do not aspire to the kind of high productivity that has been the dependent variable in this study. A different operationalisation of the efficiency and effectiveness of onboarding may be required in those cases.

Internal validity refers to the degree to which confounding factors are controlled for. The factors that we have analysed cannot be considered to constitute all the possible factors that may influence onboarding. Since all the developers in the supported condition are enrolled in university education, their educational background and other demographic factors may play a role in their performance. Also, since some degree of selection was applied, the students may represent a sample of developers with above-average knowledge and abilities. The cause-effect relationship between the characteristics considered and the efficiency and effectiveness of an onboarding process would require examining each characteristic separately in order to explain the particular impact that each of them has for specific aspects of onboarding.

Internal validity is also potentially threatened by the use of student subjects. Student behaviour may be motivated by factors that are different from those found among other kinds of developers. For instance, students may be motivated primarily by the desire to complete their degree. The lack of economic incentive among students may or may not be considered a threat to validity, depending on the viewpoint. On one hand, students may have similar motivations as volunteer OSS developers, who approach OSS development with intrinsic motives such as interest in the development work itself, or the desire to learn. On the other hand, many OSS developers are employed to work on their projects and thus may have a financial motivation as well, which the students lack. Although many newcomers in projects may be students, this cannot be considered to be the rule. There is also reason to believe that there are differences between students and experienced developers due to the accumulated skills of the latter. Both software developers who have general experience with software development and those who also have specific experience with OSS are likely to benefit from mentoring in order to quickly become oriented with the technical aspects of the software, such as source code structure and software architecture – the mentor may be able to provide helpful hints or connect the developer with more knowledgeable colleagues. However, developers who have little experience with OSS likely need more advice and guidance on project procedures, integration of tools into the development process, and project culture. Thus the exact benefits are likely to differ, and the balance of mentoring activities may need to be adjusted to benefit developers with differing levels of experience.

External validity represents the degree to which findings can be generalised, and the extent to which the findings are useful beyond the cases investigated. Although we have shown that the efficiency and effectiveness of onboarding can be influenced to some extent by mentoring and project characteristics, we have not analysed these factors in great detail, which limits our ability to determine which contextual considerations exist for generalisation. Although we expect the research questions to be of general interest, we have not shown how the desirable features of mentoring affect specific aspects of the onboarding process. This may limit the usefulness of this study.

Finally, we consider the reliability of this study to be high. As it relies mostly on publicly available quantitative data, it is highly replicable, and the results of the quantitative analysis methods are researcher-independent. However, the measures of activity rely on public communication; untracked, private communication is not directly visible in the study. Despite this limitation, such communication is publicly visible if it has resulted in actual work.

8. CONCLUSIONS

Onboarding is a fundamental process in software development that aims to motivate newcomers and ensure that they can obtain the knowledge, skills, and behaviours necessary to function in their new organisations. As OSS projects have become more important to companies, it becomes important to consider how onboarding processes can be developed for OSS development environments.

In this paper, we have presented an empirical study that analyses factors that may influence the efficiency and effectiveness of onboarding in OSS projects. We chose to examine mentoring, as it was an important aspect of the onboarding process under study, as well as a set of project characteristics on which OSS projects may differ significantly. The project characteristics were number of commits, number of contributors, appeal, lifetime, and peripheral collaboration.

We found that developers receiving onboarding support were more efficient and effective than developers who entered the same projects through the natural, non-deliberate process which is common in many OSS projects. We also found that there were differences in onboarding results which could be attributed to OSS project and community characteristics, such as project age and distribution of contributions among project members. In particular, appeal and lifetime had an important influence on the result of onboarding. Our findings can suggest how to tailor onboarding activities for particular purposes. We offered some suggestions for companies and OSS projects considering deliberate onboarding.

We expect that more factors will be studied in the future. Also, the same factors can be considered in further detail. Furthermore, a larger and broader set of projects with different and more diverse characteristics could be used. The location of developers, specific infrastructural facilities that may influence development, the time invested by developers working on the project per day, and several other factors can be analysed. Mentoring can be studied in more detail by characterising specific kinds of mentoring practices and approaches adapted to the kind of developers and projects involved. More explicit links between particular OSS project characteristics and onboarding can be drawn, isolating their impact, and more characteristics may be revealed to be important as a larger number of projects is considered.

The time window considered might be enlarged to consider the evolution of developers that have received onboarding support compared to those included by the natural process of involvement. Evaluating whether the engagement and motivation of developers is maintained after the onboarding process has concluded is highly relevant in assessing the success of the process. Likewise, an interesting question is whether peripheral developers reach the same levels of speed and contribution over time as those that have received onboarding support. If so, this would imply that both processes are equally effective, while deliberate onboarding could still be more efficient.

The collaboration program that delineated the context of our study is a good example of how companies can establish mechanisms that help them to gather new developers into OSS projects that are in their corporate interests.

In the future, we plan to devise guidelines that serve to get developers successfully onboard in OSS projects, with mechanisms and practices that are adaptable to the specific characteristics of different projects. The understanding and correct characterisation of onboarding in OSS projects may also benefit the understanding of this process in other contexts and for other types of projects.

9. REFERENCES

[1] P. Abrahamsson, P. Kettunen, and F. Fagerholm. The set-up of a software engineering research infrastructure of the 2010s. In *Proceedings of the 11th International Conference on Product Focused Software*, pages 112–114, New York, NY, USA, 2010. ACM.

[2] A. Alali, H. Kagdi, and J. I. Maletic. What's a Typical Commit? A Characterization of Open Source Software Repositories. In *Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, ICPC '08, pages 182–191, Washington, DC, USA, 2008. IEEE Computer Society.

[3] V. Basili and D. Weiss. A Methodology for Collecting Valid Software Engineering Data. *Software Engineering, IEEE Transactions on*, SE-10(6):728–738, 1984.

[4] 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*, 92(3):707–721, 2007.

[5] T. N. Bauer and B. Erdogan. Organizational socialization: The effective onboarding of new employees. 2011.

[6] A. Begel and B. Simon. Novice software developers, all over again. In *Proceedings of the Fourth international Workshop on Computing Education Research*, ICER '08, pages 3–14, New York, NY, USA, 2008. ACM.

[7] A. Begel and B. Simon. Struggles of new college graduates in their first software development job. *SIGCSE Bull.*, 40(1):226–230, Mar. 2008.

[8] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open Borders? Immigration in Open Source Projects. In *Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on*, pages 6–6, 2007.

[9] C. Bird, P. Rigby, E. Barr, D. Hamilton, D. German, and P. Devanbu. The promises and perils of mining git. In *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, pages 1–10, 2009.

[10] A. Bonaccorsi, D. Lorenzi, M. Merito, and C. Rossi. Business Firms' Engagement in Community Projects. Empirical Evidence and Further Developments of the Research. In *Emerging Trends in FLOSS Research and Development, 2007. FLOSS '07. First International Workshop on*, pages 13–13, 2007.

[11] A. Capiluppi and D. Izquierdo-Cortázar. Effort estimation of FLOSS projects: a study of the Linux kernel. *Empirical Software Engineering*, 18(1):60–88, 2013.

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

[13] K. Crowston, H. Annabi, J. Howison, and C. Masango. Effective work practices for software engineering: free/libre open source software development. In *Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research*, WISER '04, pages 18–26, New York, NY, USA, 2004. ACM.

[14] K. Crowston and J. Howison. Assessing the health of open source communities. *Computer*, 39(5):89–91, 2006.

[15] 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 Comput. Surv.*, 44(2):7:1–7:35, Mar. 2008.

[16] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1277–1286, New York, NY, USA, 2012. ACM.

[17] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 275–284, New York, NY, USA, 2010. ACM.

[18] N. Ducheneaut. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Comput. Supported Coop. Work*, 14(4):323–368, Aug. 2005.

[19] S. Elliott Sim and R. C. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In *Proceedings of the 20th international conference on Software engineering*, ICSE '98, pages 361–370, Washington, DC, USA, 1998. IEEE Computer Society.

[20] F. Fagerholm, A. S. Guinea, J. Münch, and J. Borenstein. Dataset for the paper: The Role of Mentoring and Project Characteristics for Onboarding in Open Source Software Projects. figshare.

`http://dx.doi.org/10.6084/m9.figshare.971155`. Retrieved Mar 24, 2014.

[21] F. Fagerholm, P. Johnson, A. S. Guinea, J. Borenstein, and J. Münch. Onboarding in Open Source Software Projects: A Preliminary Analysis. *International Conference on Global Software Engineering*, 2013. in press.

[22] F. Fagerholm, N. Oza, and J. Münch. A Platform for Teaching Applied Distributed Software Development: The Ongoing Journey of the Helsinki Software Factory. *Collaborative Teaching of Globally Distributed Software Development*, 2013.

[23] Y. Fang and D. Neufeld. Understanding Sustained Participation in Open Source Software Projects. *J. Manage. Inf. Syst.*, 25(4):9–50, Apr. 2009.

[24] M. Frigge, D. C. Hoaglin, and B. Iglewicz. Some Implementations of the Boxplot. *The American Statistician*, 43(1):pp. 50–54, 1989.

[25] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill. A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 385–394, New York, NY, USA, 2010. ACM.

[26] C. Gini. Variabilitá et mutabilita, 1921. In E. Pizetti and T. Salvemini, editors, *Memorie di metodologia statistica*. Libreria Eredi Virgilio Veschi, Rome, 1955. Reprinted.

[27] G. Gousios, E. Kalliamvakou, and D. Spinellis. Measuring developer contribution from software repository data. In *Proceedings of the 2008 international working conference on Mining software repositories*, MSR '08, pages 129–132, New York, NY, USA, 2008. ACM.

[28] S. Grand, G. Von Krogh, D. Leonard, and W. Swap. Resource allocation beyond firm boundaries: A multi-level model for open source innovation. *Long Range Planning*, 37(6):591–610, 2004.

[29] L. Hemphill and A. Begel. Not Seen and Not Heard. Technical report, Microsoft, 2011.

[30] J. Howison and K. Crowston. The perils and pitfalls of mining SourceForge. *26th International Conference on Software Engineering - W17S Workshop "International Workshop on Mining Software Repositories (MSR 2004)"*, 4:7–11, 2004.

[31] S.-K. Huang and K.-m. Liu. Mining version histories to verify the learning process of legitimate peripheral participants. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[32] C. Jergensen, A. Sarma, and P. Wagstrom. The onion patch: migration in open source ecosystems. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 70–80, New York, NY, USA, 2011. ACM.

[33] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346, July 2002.

[34] S. Oreg and O. Nov. Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Comput. Hum. Behav.*, 24(5):2055–2073, Sept. 2008.

[35] P. Runeson, M. Höst, A. Rainer, and B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2012.

[36] A. Schilling, S. Laumer, and T. Weitzel. Train and retain: the impact of mentoring on the retention of FLOSS developers. In *Proceedings of the 50th annual conference on Computers and People Research*, SIGMIS-CPR '12, pages 79–84, New York, NY, USA, 2012. ACM.

[37] B. Shibuya and T. Tamai. Understanding the process of participating in open source communities. In *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, FLOSS '09, pages 1–6, Washington, DC, USA, 2009. IEEE Computer Society.

[38] S. Sowe, I. Stamelos, and L. Angelis. Identifying knowledge brokers that yield software engineering knowledge in OSS projects. *Information and Software Technology*, 48(11):1025–1033, 2006.

[39] S. K. Sowe, I. Stamelos, and L. Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *J. Syst. Softw.*, 81(3):431–446, Mar. 2008.

[40] W. Swap, D. Leonard, M. Shields, and L. Abrahams. Using Mentoring and Storytelling to Transfer Knowledge in the Workplace. *J. Manage. Inf. Syst.*, 18(1):95–114, May 2001.

[41] The Debian Project. Debian New Members Corner. `http://www.debian.org/devel/join/newmaint`. Retrieved Mar 24, 2014.

[42] G. von Krogh and S. Spaeth. The open source software phenomenon: Characteristics that promote research. *The Journal of Strategic Information Systems*, 16(3):236–253, 2007.

[43] G. von Krogh, S. Spaeth, and S. Haefliger. Knowledge Reuse in Open Source Software: An Exploratory Study of 15 Open Source Projects. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 198b–198b, 2005.

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

[45] C. Wohlin, M. Höst, and K. Henningsson. Empirical Research Methods in Software Engineering. In R. Conradi and A. Wang, editors, *Empirical Methods and Studies in Software Engineering*, volume 2765 of *Lecture Notes in Computer Science*, pages 7–23. Springer Berlin Heidelberg, 2003.

[46] M. Zhou and A. Mockus. Developer fluency: achieving true mastery in software projects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pages 137–146, New York, NY, USA, 2010. ACM.